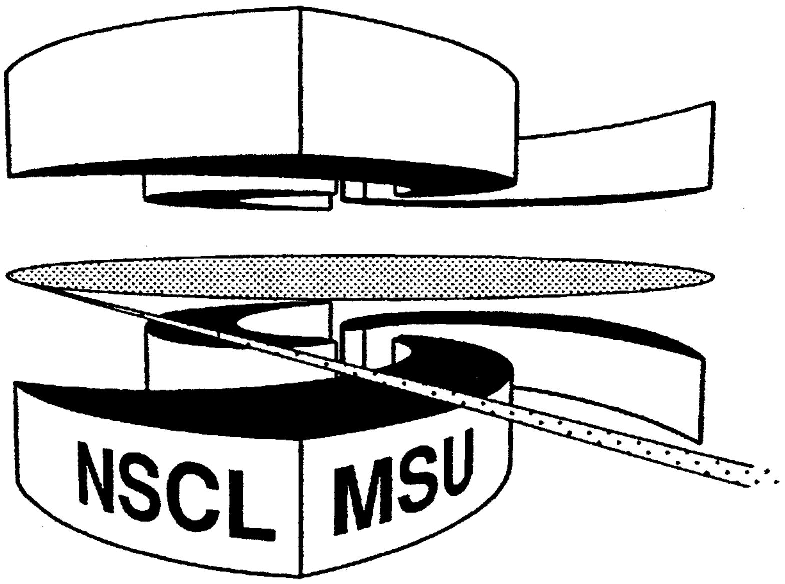
COSY INFINITY Version 8 User's Guide and Reference Manual



MSUCL-1173

JANUARY 2000

COSY INFINITY Version 8 User's Guide and Reference Manual [*]

M. Berz
Department of Physics and Astronomy
and National Superconducting
Cyclotron Laboratory
Michigan State University
East Lansing, MI 48824

January 28, 2000

Abstract

This is a reference manual for the arbitrary order beam physics code COSY INFINITY. It is current as of January 28, 2000. COSY INFINITY is a powerful new generation code for the study and design of beam physics systems including accelerators, spectrometers, beamlines, electron microscopes, and glass optical systems. At its core it is using differential algebraic (DA) methods, which allow a systematic calculation of arbitrary order effects of arbitrary particle optical elements. At the same time, it allows the computation of dependences on system parameters, which is often important and can also be used for fitting.

COSY INFINITY has a full structured object oriented language environment. This provides a simple interface for the casual user. At the same time, it offers the demanding user a very flexible and powerful tool for the study and design of systems. Elaborate optimizations are easily customized to the problem. The inclusion of new particle optical elements is straightforward.

COSY INFINITY provides a powerful environment for efficient utilization of DA techniques. The power and generality of the environment is perhaps best demonstrated by the fact that all the physics routines of COSY INFINITY are written in its own input language.

Altogether, the uniqueness of COSY lies in the ability to handle high order maps, and the ability to compute maps of arbitrary systems. Furthermore, its powerful work environment adopts to any conceivable problem. Its interactive, flexible graphics helps visualize even complicated connections between quantities.

[*]Supported in part by the U.S. Department of Energy and the Alfred P. Sloan Foundation.

Contents

1 Before Using COSY INFINITY

1.1 User's Agreement

COSY INFINITY can be obtained from M. Berz under the following conditions.

Users are requested not to make the code available to others, but ask them to obtain it from us. We maintain a list of users to be able to send out regular updates, which will also include features supplied by other users.

The FORTRAN portions and the high-level COSY language portions of the code should not be modified without our consent. This does not include the addition of new optimizers and new graphics drivers as discussed in sections 7.1 and 7.2; however, we would like to receive copies of new routines for possible inclusion in the master version of the code.

Though we do our best to keep the code free of errors and hope that it is so now, we do not mind being convinced of the contrary and ask users to report any errors. Users are also encouraged to make suggestions for upgrades, or send us their tools written in the COSY language.

If the user thinks the code has been useful, we would like to see this acknowledged by referencing some of the papers related to the code, for example [1, 2]. Finally, we do neither guarantee correctness nor usefulness of this code, and we are not liable for any damage, material or emotional, that results from its use.

By using the code COSY INFINITY, users agree to be bound by the above conditions.

1.2 How to Obtain Help and to Give Feedback

While this manual is intended to describe the use of the code as completely as possible, there will most likely arise questions that this manual cannot answer. Furthermore, we encourage users to contact us with any suggestions, criticism, praise, or other feedback they may have. We also appreciate receiving COSY source code for utilities users have written and find helpful.

We prefer to communicate by www or electronic mail. We can be contacted as follows:

Prof. Martin Berz
Department of Physics and Astronomy
Michigan State University
East Lansing, MI 48824, USA
Phone: 517-333-6313

FAX: 517-353-5967
email: berz@pa.msu.edu
http://cosy.nscl.msu.edu

1.3 How to Install the Code

The code for COSY INFINITY consists of the following files:

- FOXY.FOP
- DAFOX.FOP
- FOXFIT.FOP
- FOXGRAF.FOP
- COSY.FOX

All the system files of COSY INFINITY are currently distributed via the web; http://cosy.nscl.msu.edu/.

Four files, FOXY.FOP, DAFOX.FOP, FOXFIT.FOP and FOXGRAF.FOP, are written in FORTRAN and have to be compiled and linked. FOXY.FOP is the compiler and executer of the COSY language. DAFOX.FOP contains the routines to perform operations with objects, in particular the differential algebraic routines. FOXFIT.FOP contains the package of nonlinear optimizers. FOXGRAF.FOP contains the available graphics output drivers, which are listed in subsection 7.2.2 beginning on page 70.

In FOXGRAF.FOP, the PGPLOT and the GKS graphics driver routines are contained as standard graphics output in COSY INFINITY. The PGPLOT graphics library is freely available from the web page http://astro.caltech.edu/~tjp/pgplot/, and can be installed to VMS, UNIX, Windows 95/98/NT etc. See page 71 for more information and an example makefile for a Linux system in page 8. If not desired, the GKS or/and PGPLOT driver routines in FOXGRAF.FOP should be removed and replaced by the provided dummy routines. Some of the other popular graphics drivers, direct PostScript output and direct LaTeX output, are self contained in FOXGRAF.FOP and don't require to link to other libraries.

COSY.FOX contains all the physics of COSY INFINITY, and is written in COSY INFINITY's own input language. It has to be compiled by FOXY as part of the installation process. For this purpose, FOXY has to be run with the input file COSY.

All the FORTRAN parts of COSY INFINITY are written in standard ANSI FORTRAN 77. However, certain aspects of FORTRAN 77 are still system dependent; in particular, this concerns file handling. All system dependent features of COSY INFINITY are coded for various machines, including VAX/VMS, Windows PC, UNIX, Linux, HP, IBM mainframes, and CRAY (HP, IBM mainframes, CRAY are not actively maintained at this time).

The type of machine can be changed by selectively adding and removing comment identifiers from certain lines. To go from VAX to UNIX, for example, all lines that have the identifier *VAX somewhere in columns 73 through 80 have to be commented, and all lines that have the comment *UNIX in columns 1 through 5 have to be un-commented. To automatize this process, there is a utility FORTRAN program called VERSION that performs all these changes automatically. Should there be additional problems, a short message to us would be appreciated in order to facilitate life for future users on the same system.

1.3.1 VAX/Open VMS systems

The FORTRAN source is by default compatible with VAX/Open VMS systems. Compilation should be done without any options. In order to link and run the code, it may be necessary to increase certain working set parameters. The following parameters, generated with the VAX/Open VMS command SHOW WORK, are sufficient:

```
Working Set (pagelets) /Limit=1408  /Quota=10240  /Extent=128000
Adjustment enabled     Authorized Quota=10240  Authorized Extent=128000

Working Set (8Kb pages) /Limit=88  /Quota=640  /Extent=8000
                       Authorized Quota=640  Authorized Extent=8000
```

If PGPLOT graphics is desired, the code has to be linked with the local PGPLOT libraries.

If GKS graphics is desired, the code has to be linked with the local GKS object code. It can be executed on workstations with UIS graphics, with Xwindows graphics, and on terminals supporting Tektronix.

1.3.2 Windows PC

An executable program for Microsoft Windows 95/98/NT by the DIGITAL Visual Fortran compiler 5.0 linked with the PGPLOT graphics libraries is available.

In case compilation and linking on local machines are needed, the four FORTRAN source files have to be adjusted; all lines that contain the string *PC in columns 1 to 3 should be un-commented, and all the lines containing the string *VAX in columns 73 to 80 should be commented. This can be done using the small program VERSION; at first, adjust VERSION manually so it performs file handling properly.

If PGPLOT graphics is desired, the code has to be linked with the local PGPLOT libraries.

If VGA graphics packages with Lahey F77/F90 compilers are desired, FOXGRAF.FOP has to be adjusted; see page 72.

1.3.3 Standard UNIX systems

On Standard UNIX systems, all lines that contain the string *UNIX in columns 1 to 5 should be un-commented, and all the lines containing the string *VAX in columns 73 to 80 should be commented. This can be done using the small program VERSION; at first, adjust VERSION manually so it performs file handling properly.

On SunOS/Solaris systems, compilation should be performed with the compiler option "-Bstatic".

If PGPLOT graphics is desired, the code has to be linked with the local PGPLOT libraries.

If GKS graphics is desired, the code has to be linked to the local GKS object code. On workstations, the graphics can be utilized under Xwindows and Tektronix.

1.3.4 G77 systems (Linux)

On systems that use the GNU Fortran 77 compiler g77 and the appropriate GNU libraries, all lines that contain the string *G77 in columns 1 to 4 should be un-commented, and all the lines containing the string *VAX in columns 73 to 80 should be commented. This can be done using the small program VERSION; at first, adjust VERSION manually so it performs file handling properly.

The following is an example "Makefile" to compile and link the program with the PGPLOT graphics libraries. Check the documentation of the GNU Fortran 77 compiler about platform specific options.

```
FC=g77 -Wall
OPTFLAGS :=-O2
LIBS=-L/usr/local/pgplot -lpgplot -L/usr/X11R6/lib -lX11
OBJ = dafox.o foxy.o foxfit.o foxgraf.o

all: $(OBJ)
	$(FC) -o cosy $(OBJ) $(LIBS)

dafox.o: dafox.f
	$(FC) $(OPTFLAGS) -c dafox.f   -o dafox.o
```

```
foxy.o: foxy.f
        $(FC) $(OPTFLAGS) -c foxy.f    -o foxy.o
foxfit.o: foxfit.f
        $(FC) $(OPTFLAGS) -c foxfit.f  -o foxfit.o
foxgraf.o: foxgraf.f
        $(FC) $(OPTFLAGS) -c foxgraf.f -o foxgraf.o
```

1.3.5 HP systems

On HP systems, all lines that contain the string *HP in columns 1 to 3 should be un-commented, and all the lines containing the string *VAX in columns 73 to 80 should be commented. This can be done using the small program VERSION; at first, adjust VERSION manually so it performs file handling properly.

Compilation should be performed with the compiler option setting static memory handling.

If PGPLOT graphics is desired, the code has to be linked with the local PGPLOT libraries.

If GKS graphics is desired, the code should be linked to the local GKS object code. GKS on HP systems usually requires the use of INCLUDE files in the beginning of FOXGRAF.FOP as well as in all subroutines. These INCLUDE statements are contained in the HP version, but they have to be moved from column 6 to column 1, and possibly the address of the libraries has to be changed.On workstations, the graphics can be utilized under Xwindows and Tektronix.

The last versions of COSY INFINITY have not been explicitly tested on HP systems. Additional changes may be necessary.

1.3.6 IBM Mainframes

On IBM mainframe systems, all lines that contain the string *IBM in columns 1 to 4 should be un-commented, and all the lines containing the string *VAX in columns 73 to 80 should be commented. This can be done using the small program VERSION; at first, adjust VERSION manually so it performs file handling properly.

The last versions of COSY INFINITY have not been explicitly tested on IBM Mainframes. Additional changes may be necessary.

1.3.7 CRAY

The installation to CRAY machines with UNIX operating systems should follow the instruction in subsection 1.3.3 on Standard UNIX systems.

On CRAY machines with the original CRAY operating systems, all lines that contain the string *CRAY in columns 1 to 5 should be un-commented, and all the lines containing the string *VAX in columns 73 to 80 should be commented. This can be done using the small program VERSION; at first, adjust VERSION manually so it performs file handling properly.

The last versions of COSY INFINITY have not been explicitly tested on CRAYs. Additional changes may be necessary.

1.3.8 Possible Memory Limitations

Being based on FORTRAN, which does not allow dynamic memory allocation, COSY INFINITY has its own memory management within a large FORTRAN COMMON block. On machines supporting virtual memory, the size of this block should not present any problem. On some other machines, it may be necessary to scale down the length. This can be achieved by changing the parameter LMEM at all occurrences in FOXY.FOP, DAFOX.FOP and FOXGRAF.FOP to a lower value. Values of around 500 000 should be enough for many applications, which brings total system memory down to about 8 Megabytes.

In the case of limited memory resources, it may also be necessary to scale down the lengths of certain variables in COSY.FOX to lower levels. In particular, this holds for the variables MAP and MSC which are defined at the very beginning of COSY.FOX. Possible values for the length are values down to about 500 for work through around fifth order. For higher orders, larger values are needed.

1.4 How to Avoid Reading This Manual

The input of COSY INFINITY is based on a programming language which is described in detail in section 6 beginning on page 58. The structure and features are quite intuitive, and we are confident that one can quickly pick up the key ideas following some examples.

COSY INFINITY is written in this language, and all particle optical elements and control features are invoked by calls to library procedures written in this language. A detailed description of these features is provided in sections 3 and 4.

Section 5 beginning on page 48 gives several examples for problems occurring in the computation and analysis of particle optical systems. Reading these sections should enable the user to get a head start in using COSY INFINITY. Another source of information is the demonstration file DEMO.FOX.

For sophisticated problems or the development of customized features, the user may find it helpful to study section 7 beginning on page 67. The appendix beginning on page 78 contains a complete list of all data types and operations as well as all intrinsic functions and procedures available in the COSY language. Finally, the pages of the listing of COSY INFINITY can be consulted for existing structures and programming ideas.

1.4.1 Syntax Changes since Version 7

With very minor exceptions, version 8 is downward compatible to the previous version, so any user deck for version 7 should run under version 8.

2 What is COSY INFINITY

The design and analysis of particle optical systems is quite intimately connected with the computer world. There are numerous more or less widespread codes for the simulation of particle optical systems. Generally, these codes fall into two categories. One category includes ray tracing codes which use numerical integrators to determine the trajectories of individual rays through external and possibly internal electromagnetic fields. The core of such a code is quite robust and easy to set up; for many applications, however, certain important information can not be directly extracted from the mere values ray coordinates. Furthermore, this type of code is often quite slow and does not allow extensive optimization.

The other category of codes are the map codes, which compute Taylor expansions to describe the action of the system on phase space. These codes are usually faster than integration codes, and the expansion coefficients often provide more insight into the system. On the other hand, in the past the orders of the map, which are a measure of the accuracy of the approach, have been limited to third order [3, 4, 5] and fifth order [6, 7]. Furthermore, traditional mapping codes have only very limited libraries for quite standardized external fields and lack the flexibility of the numerical integration techniques. In particular, fringe fields can only be treated approximately.

2.1 COSY's Algorithms and their Implementation

Recently we could show that it is indeed possible to have the best of both worlds: using the new differential algebraic techniques, any given numerical integration code can be modified such that it allows the computation of Taylor maps for arbitrarily complicated fields and to arbitrary order [8, 9, 10, 11]. An offspring of this approach is the computation of maps for large accelerators where often the system can be described by inexpensive, low order kick integrators [12, 13].

The speed of this approach is initially determined by the numerical integration process. Recently it has been possible to use DA techniques to overcome this problem:

DA can be used to automatically emulate numerical integrators of very high orders in the time step, yet at the computational expense of only little more than a first order integrator [8, 9]. This technique is very versatile, works for a very large class of fields, and the speeds obtained are similar to classical mapping codes.

In order to make efficient use of DA operations in a computer environment, it has to be possible to invoke the DA operations from a language. In the conventional languages used for numerical applications it is not possible to introduce new data types and operations on them. Only recently have object oriented languages been developed which routinely have such features. One such language which is slowly gaining ground is C++; FORTH is another example which has been around for a longer time.

There are strong reasons to stay within the limits of a FORTRAN environment, however. Firstly, virtually all software in this field is written in this language, and the desire to interface to such software is easier if FORTRAN is used. Furthermore, there are extensive libraries of support software which are only slowly becoming available in other languages, including routines for nonlinear optimization and various graphics packages. Finally, the necessity for portability is another strong argument for FORTRAN; virtually every machine that is used for numerical applications, starting from personal computers, continuing through the workstation and mainframe world to the supercomputers, has a FORTRAN compiler.

So it seemed natural to stay within this world, and this lead to the development of the DA precompiler [14]. This precompiler allows the use of a DA data type within otherwise standard FORTRAN by transforming arithmetic operations containing DA variables into a sequence of calls to subroutines. This technique has been extensively used [9, 10, 15, 16, 17, 18]. It was particularly helpful that one could use old FORTRAN tracking codes and just replace the appropriate real variables by DA variables to very quickly obtain high order maps.

2.2 The User Interface

On the other end of the problems using an accelerator code is the command language of the code and the description of the beamlines. Various approaches have been used in the past, starting from coding numbers as in the old versions of TRANSPORT [3] over more easily readable command structures like in TRIO [4], GIOS [5, 19], COSY 5.0 [6, 16] and MARYLIE [20] to the standardized commands of MAD, for which there is a conversion utility to COSY (see section 3.4.1) [21, 22].

COSY INFINITY approaches this problem by offering the user a full programming language; in fact, the language is so powerful that all the physics of COSY INFINITY was written in it, and the results fit on a few pages.

For ease of use such a language should have a simple syntax. For the user demanding special-purpose features, it should be powerful. It should allow direct and complex interfacing to FORTRAN routines, and it should allow the use of DA as a type. Finally, it should be widely portable. Unfortunately, there is no language readily available that fulfills all these requirements, so COSY INFINITY contains its own language system.

The problem of simplicity yet power has been quite elegantly solved by the PASCAL concept. In addition, this concept allows compilation in one pass and no linking is required. This facilitates the connection of the user input, which will turn out to be just the last procedure of the system, with the optics program itself.

To be machine independent, the output of the compilation is not machine code but rather an intermediate code that can be easily interpreted. For the same reason, it is essential to write the source code of the compiler in a very portable language. We chose FORTRAN for the compiler, even though clearly it is considerably easier to write it in a recursive language.

For reasons of speed it is helpful to allow the splitting of the program into pieces, one containing the optics program and one the user commands. While the PASCAL philosophy does not have provisions for linking, it allows the splitting of the input at any point. For this purpose, a complete momentary image of the compilation status is written to a file. When compilation continues with the second portion, this image is read from the file, and compilation continues in exactly the same way as without the splitting.

The full syntax of the COSY language is described in detail in section 6 beginning on page 58. Most of the syntax will become apparent from the detailed examples supplied in the following sections, and we think that it is possible to write most COSY inputs without explicitly consulting the language reference.

3 Computing Systems with COSY

This section describes some core features of COSY's particle optics and accelerator physics environment. This provides the backbone for practical use in particle optics. We assume that the reader has a fundamental knowledge about particle optics, and refer to the literature, for example [23, 24, 25, 26, 27, 28].

3.1 General Properties of the COSY Language Environment

The physics part of COSY INFINITY is written in its own input language. In this context, most commands are just calls to previously defined procedures. If desired, the user can create new commands simply by defining procedures of his own. All commands within COSY INFINITY consist of two or three letters which are abbreviations for two or three words describing the action of the procedure. This idea originated in the GIOS language [5, 19], and many commands of COSY INFINITY are similar to respective commands in GIOS. All units used in the physics part of COSY are SI, except for voltages, which are in kV, and angles, which are in degrees.

Particle optical systems and beamlines are described by a sequence of calls to procedures representing individual elements. The supported particle optical elements can be found in section 3.3 beginning on page 22; section 5.7 beginning on page 55 shows how to generate new particle optical elements.

In a similar way, elements can be grouped, which is described in section 5.3 beginning on page 50. Besides the commands describing particle optical elements, there are commands to instruct the code what to do.

3.2 Control Commands

All user commands for COSY INFINITY are contained in a file which is compiled by FOXY. The first command of the file must be

INCLUDE 'COSY' ;

which makes all the compiled code contained in COSY.FOX known to the user input. The user input itself is contained in the COSY procedure RUN. Following the syntax of the COSY language described in section 6, all commands thus have to be included between the statements

PROCEDURE RUN ;

and

ENDPROCEDURE ;

In order to execute the commands, the ENDPROCEDURE statement has to be followed by the call to the procedure,

RUN ;

and the command to complete the COSY input file,

END ;

Like any language, the COSY language supports the use of variables and expressions which often simplifies the description of the system. For the declaration of variables, see section 6.

The first command sets up the DA tools and has to be called before any DA operations, including the computation of maps, can be executed. The command has the form

OV <order> <phase space dimension> <number of parameters> ;

and the parameters are the maximum order that is to occur as well as the dimensionality of phase space (1,2 or 3) and the number of system parameters that are requested. If the phase space dimensionality is 1, only the x-a motion is computed; if it is 2, the y-b motion is computed as well, obviously at a slightly higher computation time. If it is 3, the time of flight and chromatic effects are computed also.

The number of parameters is the number of additional quantities besides the phase space variables that the final map shall depend on. This is used in connection with the "maps with knobs" discussed in section 5.2 on page 49 and to obtain mass and charge dependences if desired, and it is also possible to compute energy dependence without time-of-flight terms at a reduced computational expense.

The order is arbitrary and denotes the maximum order that computations can be performed in. It is possible to change the computation order at run time using the command

CO <order> ;

however, the new order can never exceed the one set in **OV**. Note that the computation time naturally increases drastically for higher orders. Under normal circumstances, orders should not exceed ten very much.

3.2.1 The Coordinates

COSY INFINITY performs all its calculations in the following scaled coordinates:

$$\begin{array}{rclrcl} r_1 &=& x, & r_2 &=& a = p_x/p_0, \\ r_3 &=& y, & r_4 &=& b = p_y/p_0, \\ r_5 &=& l = -(t-t_0)v_0\gamma/(1+\gamma) & r_6 &=& \delta_K = (K-K_0)/K_0 \\ r_7 &=& \delta_m = (m-m_0)/m_0 & r_8 &=& \delta_z = (z-z_0)/z_0 \end{array}$$

The first six variables form three canonically conjugate pairs in which the map is symplectic. The units of the positions x and y is meters. p_0, K_0, v_0, t_0 and γ are the momentum, kinetic energy, velocity, time of flight, and total energy over m_0c^2, respectively. m and z denote mass and charge, respectively.

3.2.2 Defining the Beam

All particle optical coordinates are relative to a reference particle which can be defined with the command

RP <kinetic energy in MeV> <mass in amu> <charge in units> ;

For convenience, there are two procedures that set the reference particle to be protons or electrons:

RPP <kinetic energy in MeV> ;

RPE <kinetic energy in MeV> ;

For the masses of the proton and electron and all other quantities in COSY, the values in [29] have been used. Finally, there is a command that allows to set the reference particle from the magnetic rigidity in Tesla meters and the momentum in GeV/c:

RPR <magnetic rigidity in Tm> <mass in amu> <charge in units> ;

RPM <momentum in MeV/c> <mass in amu> <charge in units> ;

Finally it is possible to set the magnetic moments of the particle and activate the computation of spin . This is achieved with the command

RPS < LS > < G > ;

where LS is the spin mode, 1 indicating spin computation and 0 indicating no spin computation. $G = (g-2)/2$ is the anomalous spin factor of the particle under consideration. In case the reference particle has been set to be a proton using **RPP** or an electron using **RPE**, the proper value will be used if G is set to zero.

The command

SB <PX><PA><r12><PY><PB><r34>< PT><PD><r56><PG><PZ> ;

sets half widths of the beam in the x, a, y, b, t, d, g and z directions of phase space as well as the off diagonal terms of the ellipse in TRANSPORT notation r12, r34, and r56. The units are meters for PX and PY, radians for PA and PB, $v_0\gamma/(1+\gamma)$ times time for PT, and $\Delta E/E$ for PD, $\Delta m/m$ for PG, and $\Delta z/z$ for PZ. The command

SP <P1> <P2> <P3> <P4> <P5> <P6> ;

sets the maxima of up to six parameters that can be used as knobs in maps (see section 5.2 beginning on page 49).

SBE <EX> <EY> <ET> ;

sets the ellipse of the beam to an invariant ellipse of the current map. The emittances in x-a, y-b, and τ-δ space being <EX>, <EY>, <ET> respectively.

3.2.3 The Computation of Maps

COSY INFINITY has a global variable called MAP that contains the accumulated transfer map of the system. Each particle optical element being invoked updates the momentary contents of this global variable.

The following command is used to prepare the computation of maps. It sets the transfer map to unity. It can also be used again later to re-initialize the map.

UM ;

The command

SM <name> ;

saves the momentary transfer matrix to the array name, which has to be specified by the user. The array can be specified using the **VARIABLE** command of the COSY language (see section 6). It could have the form

VARIABLE <name> 1000 8 ;

which declares a one dimensional array with eight entries. Each entry can hold a maximum of 1000 16 byte blocks, which should be enough to store the DA numbers occurring in calculations of at least seventh order.

To copy a map stored in an array name1 to another array name2, use the procedure

SNM <name1> <name2> ;

The command

AM <name> ;

applies the previously saved map <name> to the momentary map. AM and PM are particularly helpful for the handling of maps of subsystems that are expensive to calculate. In particular in the context of optimization, often substantial amounts of time can be saved by computing certain maps only once and then re-using them during the optimization.

It is also sometimes necessary to compose two individual maps into one map without acting on the current transfer map. This can be achieved with the command

ANM <N> <M> <O> ;

which composes the maps N and M to O=N $\circ$ M. The command

PM <unit> ;

prints the momentary transfer matrix to unit. This number can be associated with a file name with the OPENF procedure (see index); if OPENF is not used, the name associated with the unit follows the local FORTRAN conventions. Unit 6 corresponds to the screen. The different columns of the output belong to the final values of x, a, y, b and t of the map, and different lines describe different coefficients of the expansion in terms of initial values. The coefficients are identified by the last columns which describe the order as well as the exponents of the initial values of the variables. An example of the output of a transfer map can be found in section 5 on page 48.

The command

PSM <unit> ;

writes the 3×3 spin matrix to unit.

Besides the easily legible form of output of a transfer map produced by **PM**, it is also possible to write the map more accurately but less readable with the command

WM <unit> ; .

In this case, the transformation of the local coordinate system is also stored and can be reused when read. Maps written by **PM** or **WM** can be read with the command

RM <unit> ;

reads a map generated by **PM** from the specified unit and applies it to the momentary transfer map. Often a significant amount of computer time can be saved by computing certain submaps ahead of time and storing them either in a variable or a file. In particular this holds for maps which are expensive to compute, for example the ones of electrostatic cylindrical lenses.

Besides storing maps of an element or system with one specific setting of parameters, using the technique of symplectic scaling it is possible to save maps with a certain setting of filed strengths and lengths and later re–use them for different settings of lengths or strengths. This is particularly useful for elements that require a lot of calculation time, including fringe fields and solenoids. A representation of the map of an element with typical dimensions and field strength for a typical beam is saved using

WSM < unit> <L> <B> <D> ;

This map has to be calculated either in three dimensions (OV order 3 0 ;) or with the energy as a parameter (OV order 2 1 ;). The parameters are the output unit, length, pole–tip field, and aperture of the element that created the momentary map. The map of the motion of a different type of beam through any similar element that differs in scale or field strength can be approximated quickly by

RSM <unit> <L> <B> <D> ;

It is also possible to extract individual matrix elements of transfer maps. This is achieved with the COSY function

ME (<phase space variable>,<element identifier>)

The element identifier follows TRANSPORT notation; for example, ME(1,122) returns the momentary value of the matrix element (x, xaa).

The beam's current sigma matrix is computed from the ellipse data previously set with SB by the function

SIGMA (<I>,<J>)

Sometimes it is necessary to determine the map of the reversed system, i.e. the system transversed backwards. In case M is the map of the system, the map MR of the corresponding reversed system can be computed with the command

MR <M> <MR> ;

Note again that the current transfer map is stored in the global variable MAP. Similarly, it is sometimes necessary to determine the map of the system in which the coordinates are twisted by a certain angle. For example, if the direction of bending of all magnets is exchanged, this corresponds to a rotation by 180 degrees. In case M is the map of the system, the map MT of the system twisted by angle can be computed with the command

MT <M> <MT> <angle> ;

3.2.4 The Computation of Trajectories

Besides the computation of maps, COSY can also trace rays through the system. The trajectories of these rays can be plotted or their coordinates printed. If rays are selected, they are pushed through every new particle element that is invoked. Note that COSY can also push rays through maps repetitively and display phase space plots. This uses different methods and is discussed in section 4.4 beginning on page 44.

The following command sets a ray that is to be traced through the system. The parameters are the eight particle optical coordinates

SR <X> <A> <Y> <B> <T> <D> <G> <Z> <color> ;

Here X and Y are the positions of the ray in meters, A and B are the angles in radians, T is the time of flight multiplied by $v_0\gamma/(1+\gamma)$. D, G and Z are the half energy, mass and charge deviations. For graphics purposes, it is also possible to assign a color. Different colors are represented by numbers as follows. 1: black, 2: blue, 3: red, 4: yellow, 5: green. The command

SSR <X> <Y> <Z> ;

sets the spin coordinates of the particle. Note that command has to be used immediately following the setting of the coordinates of the particle with **SR**.

It is also possible to automatically set an ensemble of rays. This can be achieved with the command

ER <NX> <NA> <NY> <NB> <NT> <ND> <NG> <NZ> ;

Here NX, NA ... denote the number of rays in the respective phase space dimension and have to be greater than or equal to 1. The ray coordinates are equally spaced according to the values set with the command **SB**, which has to be called before **ER**. In case any of the N's is 1, only rays with the respective variable equal to 0 will be shown. Note that the total number of rays is given by NX · NA ·...· NZ, which should not exceed 100. Note that this command is incompatible with the setting of spin coordinates with **SSR** as described above. The command

SCDE ;

sets sine like and cosine like rays as well as the dispersive ray and the beam envelope in accordance with the data provided by SB or SBE. After the envelope has been set by SCDE it can be displayed alone as it varies along the system with PGE, or together with the other trajectories with PG. If only the envelope should be evaluated,

ENVEL ;

should be used. The closed orbit for an off energy particle, often called the η function, is produced by

ENCL <D> ;

The periodic orbit for an off energy particle with the dispersion D is computed from the one turn map. Therefore a current map has to be produced before calling ENCL. This is equivalent to the requirement of computing a current map before calling SBE.

CR ;

clears all the rays previously set. The command

PR <unit> ;

prints the momentary coordinates of the rays to the specified unit. Unit 6 corresponds to the screen. Note that using the **WRITE** command of the COSY language, it is also possible to print any other quantity of interest either to the screen or to a file.

3.2.5 Plotting System and Trajectories

Besides computing matrices and rays, COSY also allows to plot the system or any part of it and the rays going through it. The command

PTY < scale > ;

selects the type of system plot. If scale is zero, the reference trajectory will be plotted as a straight line; this is also the default if **PTY** is not called. If scale is nonzero, all rays including the reference trajectory are displayed in laboratory coordinates. To account for the fact that in such a view rays are rather close to the reference trajectory and hence may be hard to distinguish, the coordinates transverse to the optic axis will be magnified by the value of scale.

BP ;

defines the beginning of a section of the system that is to be plotted, and the command

EP ;

defines the end of the section. The command

PP <unit> <phi> <theta> ;

plots the system to unit. Following the convention of printing graphics objects discussed in section 7.2 beginning on page 69, positive units produce a low-resolution ASCII plot of 80 columns by 24 lines, which does not require any graphics packages. Negative units correspond to various graphics standards.

The picture of the trajectories and elements is fully three dimensional and can be viewed from different angles. Phi=0 and Theta=0 correspond to the standard x projection; Phi=0 and Theta=90 correspond to the y projection; and Phi=90 and Theta=0 correspond to viewing the rays along the beam.

For use on workstations, there is also an abbreviated way to produce both an x projection and a y projection simultaneously. The command

PG <Unit1> <Unit2> ;

produces both x and y pictures, including length (lower right), height (upper left) and depth (lower left) of the system with all selected rays and the envelope if selected. Unit1 and Unit2 denote the Graphics units (see section 7.2). The command

PGE <Unit1> <Unit2> ;

produces both x and y pictures, including length (lower right), height (upper left) and depth (lower left) of the system and the beam envelope. Unit1 and Unit2 denote the Graphics units (see section 7.2).

In a picture, it is sometimes advantageous to identify a particular location on the reference trajectory, for example to identify a focal plane or a plane of interest in a ring. This can be achieved with the command

PS <d> ;

which draws an poincare section plane with width d at the momentary position of the reference trajectory.

There are several parameters which control the graphic output of a system. Such a graphic displays the central trajectory along with all rays and the envelope, the optical elements, two letters below each element indicating its type and three numbers indicating the height, width, and depth of the system. Before the system is computed, this default can be changed by

- LSYS = 0 ; (Suppresses the beamline elements) ,
- LCE = 0 ; (Suppresses the types of the elements) ,
- LAX = 0 ; (Suppresses the numbers describing the size of the system) .

These options can become important when graphic output of huge machines is desired. These choices can then avoid memory overflow and uncomprehendable picture.

3.3 Supported Elements

In this section we present a list of all elements available in COSY. They range from standard multipoles and sectors over glass lenses and electromagnetic cylindrical lenses to a general element, which allows the computation of the map of any element from measured field data. The maps of all elements can be computed to arbitrary order and with arbitrarily many parameters.

Elements based on so-called strong focusing devices like multipoles and sectors can be computed with their fringe fields or without, which is the default. Section 3.3.4 beginning on page 26 describes various fringe field computation modes available.

The simplest particle optical element, the field- and material free drift, can be applied to the map with the command

DL <length> ;

The element

CB ;

changes the bending direction of bending magnets and deflectors. Initially, the bending direction is clockwise. The procedure **CB** changes it to counterclockwise, and each additional **CB** switches it to the other direction. Note that it is also possible to change the bending direction of all the elements in an already computed map using the command **MX** (see index).

COSY supports a large ensemble of other particle optical elements, and it is very simple to add more elements. The following subsections contain a list of momentarily available elements.

3.3.1 Multipoles

COSY supports magnetic and electric multipoles in a variety of ways. There are the following magnetic multipoles:

MQ <length> <flux density at pole tip> <aperture> ;

MH <length> <flux density at pole tip> <aperture> ;

MO <length> <flux density at pole tip> <aperture> ;

MD <length> <flux density at pole tip> <aperture> ;

MZ <length> <flux density at pole tip> <aperture> ;

which let a magnetic quadrupole, sextupole, octupole, decapole or duodecapole act on the map. The aperture is the distance from reference trajectory to pole tip. For the sake of speed, direct formulas for the aberrations are used for orders up to two. There is also a superimposed multipole for multipole strengths up to order five:

M5 <length> <BQ >< BH >< BO >< BD >< BZ> <aperture> ;

And finally, there is a general superimposed magnetic multipole with arbitrary order multipoles:

MM <length> <MA> <NMA> <aperture> ;

Contrary to the previous procedure, the arguments now are the array MA and the number NMA of supplied multipole terms. Besides the magnetic multipole just introduced, which satisfies midplane symmetry, there is also a routine that allows the computation of skew multipoles. The routine

MMS <length> <MA> <MS> <NMA> <aperture> ;

lets a superposition of midplane symmetric and skew multipoles act on the map. The array MA contains the strengths of the midplane symmetric multipoles in the same units as above. The array MS contains the strengths of the skew multipoles; the units are such that a pure skew 2n pole corresponds to the midplane symmetric multipole with the same strength rotated by an angle of $\pi/2n$.

Similar procedures are available for electrostatic multipoles

EQ <length> <voltage at pole tip> <aperture> ;

EH <length> <voltage at pole tip> <aperture> ;

EO <length> <voltage at pole tip> <aperture> ;

ED <length> <voltage at pole tip> <aperture> ;

EZ <length> <voltage at pole tip> <aperture> ;

which let an electric quadrupole, sextupole, octupole, decapole or duodecapole act on the map. The strengths of the multipoles are described by their voltage in kV. There is an electric multipole

E5 <length >< EQ >< EH >< EO >< ED >< EZ> <aperture> ;

which lets a superimposed electric multipole with components EQ through EZ act on the map, and there is the procedure

EM <length> <EA> <NEA> <aperture> ;

which lets a general electrostatic multipole with arbitrary order multipoles act on the map. Similar to the magnetic case, there are also electric skew multipoles. The routine

EMS <length> <EA> <ES> <NEA> <aperture> ;

lets a superposition of midplane symmetric and skew multipoles act on the map. The array EA contains the strengths of the midplane symmetric multipoles in the same units as above. The array ES contains the strengths of the skew multipoles; like in the magnetic case, the units are such that a pure skew 2n pole corresponds to the midplane symmetric multipole with the same strength rotated by an angle of $\pi/2n$.

3.3.2 Bending Elements

COSY INFINITY supports both magnetic and electrostatic elements including so called combined function elements with superimposed multipoles. In the case of magnetic elements, edge focusing and higher order edge effects are also supported. By default, all bending elements bend the reference trajectory clockwise, which can be changed with the command **CB** (see index).

The following commands let an inhomogeneous combined function bending magnet and a combined function electrostatic deflector act on the map:

MS <radius> <angle> <aperture> $< n_1 >< n_2 >< n_3 >< n_4 >< n_5 >$;

ES <radius> <angle> <aperture> $< n_1 >< n_2 >< n_3 >< n_4 >< n_5 >$;

The radius is measured in meters, the angle in degrees, and the aperture is in meters and corresponds to half of the gap width. The indices n_i describe the midplane radial field dependence which is given by

$$F(x) = F_0 \cdot \left[1 - \sum_{i=1}^{5} n_i \cdot (\frac{x}{r_0})^i \right]$$

where r_0 is the bending radius. Note that an electric cylindrical condenser has $n_1 = 1$, $n_2 = -1$, $n_3 = 1$, $n_4 = -1$, $n_5 = 1$, etc, and an electric spherical condenser has $n_1 = 2$, $n_2 = -3$, $n_3 = 4$, $n_4 = -5$, $n_5 = 6$, etc. Homogeneous dipole magnets have $n_i = 0$.

The element

DI <radius> <angle> <aperture> $< \epsilon_1 >$ <h$_1$ > $< \epsilon_2 >$ <h$_2$ > ;

lets a homogeneous dipole with entrance edge angle ϵ_1 and entrance curvature h_1 as well as exit edge angle ϵ_2 and exit curvature h_2 act on the map. All angles are in degrees, the curvatures in 1/m, the radius is in m, and the aperture is half of the gap width. Positive edge angles correspond to weaker x focusing, and positive curvatures to weaker nonlinear x focusing.

In the sharp cut off approximation, the horizontal motion in the homogeneous dipole is based on geometry. The vertical effects of edge angle and curvatures is approximated by a linear and quadratic kick, which is a common approximation of hard-edge fringe–field effects. As described in section 3.3.4, it is also possible to treat the influence of extended fringe fields on horizontal and vertical motion in detail and full accuracy.

The element

MSS <radius r_0 > <angle ϕ_0 > <aperture> < ϵ_1 > <h_1 > < ϵ_2 > <h_2 > < w > ;

allows users to specify the two dimensional structure of the main field in polar coordinates, which is described by a two dimensional array $w_{(i,j)}$. The following factor is imposed to the main field specified by the first seven arguments with the same meaning to those of **DI**.

$$F(r,\phi) = \sum_{i=1}^{4}\sum_{j=1}^{4} w_{(i,j)}(r-r_0)^{i-1}(\phi-\phi_0/2)^{j-1}.$$

A special case of the homogeneous dipole described above is the magnetic rectangle or parallel-faced dipole, in which both edge angles equal one half of the deflection angle and the curvatures are zero. For convenience, there is a dedicated routine that lets a parallel faced magnet act on the map:

DP <radius> <angle> <aperture> ;

Finally, there is a very general combined function bending magnet with shaped entrance and exit edges

MC <radius> <angle> <aperture> <N> <S1> <S2> < n > ;

Here N is an array containing the above n_i, and S1 and S2 are arrays containing the n coefficients s_1, ... s_n of two n-th order polynomials describing the shape of the entrance and exit edges as

$$S(x) = s_1 \cdot x + ... + s_n \cdot x^n$$

Again positive zeroth order terms entail weaker x focusing. In the sharp cut off approximation, the edge effects of the combined function magnet are treated as follows. All horizontal edge effects of order up to two are treated geometrically like in the case of the dipole. The vertical motion as well as the contribution to the horizontal motion due to the non-circular edges are treated by kicks. The treatment of the element in the presence of extended fringe fields is described in section 3.3.4.

Note that when comparing COSY bending elements without extended fringe fields to those of other codes, it is important to realize that some codes actually lump some fringe–field effects into the terms of the main fields. For example, the code TRANSPORT gives nonzero values for the matrix element (x, yy), which is produced by a fringe–field effect, even if all TRANSPORT fringe–field options are turned off.

3.3.3 Wien Filters

Besides the purely magnetic and electric bending elements, there are routines for superimposed electric and magnetic deflectors, so-called Wien Filters or E cross B devices. The simplest Wien Filter consists of homogeneous electric and magnetic fields which are superimposed such that the reference trajectory is straight. This element is called by

WF <radius$_1$> <radius$_2$> <length> <aperture> ;

The radii describe the bending power of the magnetic and electric fields, respectively. The strengths are chosen such that each one of them alone would deflect the beam with the specified radius. For positive radii, the electric field bends in the direction of positive x, and the magnetic field bends in the direction of negative x. For equal radii, there is no net deflection. There is also a combined function Wien Filter:

WC <radius$_1$ > <radius$_2$ > <length> <aperture> <NE> <NM> < n > ;

Here NE and NM describe the inhomogeneity of the electric and magnetic fields, respectively via

$$F(x) = F_0 \cdot \left[1 + \sum_{i=1}^{n} N(i) \cdot x^i \right]$$

3.3.4 Fringe Fields

A detailed analysis of particle optical systems usually requires the consideration of the effects of the fringe fields of the elements. While in the default, COSY INFINITY does not take fringe fields into account, there are commands that allow the computation of their effects with varying degrees of accuracy and computational expense.

There are two ways of computing fringe fields of particle optical elements. The first way is based on the standardized description of the s-dependence of multipole strengths by an Enge function as in the program RAYTRACE [30]. The Enge function has the form

$$F(z) = \frac{1}{1 + \exp(a_1 + a_2 \cdot (z/D) + ... + a_6 \cdot (z/D)^5)}$$

where z is the distance perpendicular to the effective field boundary. In the case of multipoles, this coincides with the arc length along the reference trajectory. The quantity

D is the full aperture (in case of multipoles twice the radius) of the particle optical element, and a_1 through a_6 are the Enge coefficients.

The Enge coefficients depend on the details of the geometry of the element including shimming and saturation effects in magnetic elements. The coefficients have to be adjusted such that the Enge function fits the specific measured or computed data. There are various existing fitting programs for this purpose, one of which we believe can be obtained together with RAYTRACE [30]. Note that in the optimization process it is important that the Enge coefficients be chosen such that the effective field boundary coincides with the origin. It is also important that the fringe–field coefficients lead to an Enge function which represents the fringe field well over an interval ranging from at least six half apertures inside the element to at least ten half apertures outside the element.

While the details of the fringe–field effects depend on the exact values of the Enge coefficients a_1 through a_6 and requires their exact knowledge, in many cases the bulk of the effects can be described well with default values for the coefficients. COSY uses a set of default values representing measurements of a family of unclamped multipoles used for PEP [31]. If a higher degree of accuracy is required, it is possible to input specific Enge coefficients directly. This is achieved with the command

FC <IMP> <IEE> <IME> $< a_1 >$ $< a_2 >$ $< a_3 >$ $< a_4 >$ $< a_5 >$ $< a_6 >$;

which sets the Enge coefficients a_1 through a_6. IMP is the multipole order (1 for dipoles, 2 for quadrupoles etc). IEE identifies the data belonging to entrance (1) and exit (2) fringe fields. IME denotes magnetic (1) or electric (2) elements. Using **FC** repeatedly, it is possible to set coefficients for the description of all occurring elements. Any combination not explicitly set remains at the default.

Note that if all the elements in the system are of the same type, it is sufficient to call **FC** once for every element. In case there are different types, **FC** has to be called again before an element of a different type is executed. Sometimes it has proven helpful to lump several calls to **FC** into a procedure. One such procedure that is already part of COSY is

FD ;

which sets all values to the default; this procedure is automatically called the first time fringe fields are used and has to be called again when they should be reset to the default after changing them.

Since very detailed fringe–field calculations are sometimes rather expensive computationally, COSY allows to compute their effects with varying degrees of accuracy, which is controlled by

FR <mode> ;

If mode is 0, fringe fields are disregarded, which is the default. Mode 1 entails approximate fringe fields with an accuracy comparable to the fringe–field integral method. Mode 2 entails fringe fields with an accuracy higher yet. It uses parameter dependent symplectic map representations of fringe–field maps stored on files to approximate the fringe field via symplectic scaling [32] [33]. The default reference maps are stored in the file SYSCA.DAT . If needed, other reference files that represent the user data more closely can be created and stored on files by **WSM** (see index). How this is done can be seen in the procedure

CRSYSCA ;

This procedure produces the file SYSCA.DAT, which is shipped with the code. Such maps can be declared to be the new standard with the command

FC2 <IMP> <IEE> <file> ;

which declares file to be the actual reference file for the fringe field described by IMP and IEE; the meaning of IMP and IEE is discussed above for the command **FC**. The original default files can be reactivated by

FD2 ;

the fringe–field mode 2 is especially helpful in the final design stages of a realistic system after approximate parameters of the elements have been obtained by neglecting fringe fields or with fringe–field mode 1. The last step of the optimization can then be made using the default scaled fringe–field maps. A very high degree of accuracy almost equal to that of the fringe–field mode 3 discussed below can be obtained by computing new fringe–field reference maps with the command **WSM** based on the approximate values obtained by the previous fits.

Mode 3 finally produces the most accurate fringe fields, at a computational expense typically more than one order of magnitude higher than that of mode 2. In all cases, the mode set with FR stays effective until the next call of FR. In this case, the accuracy is limited only by the accuracy of the numerical integrator which can be set with the procedure

ESET $< \epsilon >$;

where ϵ is the maximum error in the weighted phase space norm discussed in connection with the procedure **WSET** (see index). The default for ϵ is 10^{-10} and can be adjusted downwards somewhat if needed.

It is possible to change the computation mode within the computation. Whenever a new fringe–field mode is desired, **FR** has to be called again with the new mode. This mode remains in effect until **FR** is called again.

It is also possible to calculate fringe fields of elements alone. If mode is set to -1, only entrance fringe fields of all listed elements are computed, and if mode is set to -2, only exit fringe fields are computed. In both cases, the computation is fully accurate.

These maps can be used in two ways: if the fringe fields do not change anymore, the data can stored and re-used with the commands **SM** and **AM** or also **PM** and **RM** (see section 3.2.3. In the case the maps of entrance or exit fringe fields are re-used in this way, it is important to turn all fringe fields off with the command **FR 0** because otherwise the fringe fields are taken into account twice. It is also important that in the case of bending elements with non-perpendicular entrance or exit (see section 3.3.2), the fringe–field maps computed using **FR -1** and **FR -2** do not contain the effects of any curved entrance and exit plane. In the case fringe–field maps are re-used later with turned off fringe fields, it is thus important to leave all edge effects in the body of the element.

Using modes -1 and -2, it is also possible to determine new fringe–field reference maps that can be used with symplectic scaling using the commands **WSM** and **RSM**.

Besides the computation of fringe–field effects in the formalism of Enge type multipole functions, fringe–field effects can also be computed with the general particle optical element **GE** discussed in section 3.3.8 beginning on page 33. This allows the treatment of strongly overlapping fringe fields or fringe fields that cannot be represented well by Enge functions.

In the case of straight multipole elements, in the midplane the total fringe field is the sum of the individual multipole components which fall off with their respective Enge functions. The nonlinearities of the off-plane fields are computed from this information in agreement with Maxwell's equations [1] In the case of the dipole element **DI**, the Enge function modulates the fall off of the midplane dipole field perpendicular to the edge of the magnet. As long as the edges are long enough, this allows a very accurate description both for straight and circular edges, where circular edges may require Enge coefficients that differ slightly from those of straight edges with the same aperture. Again, the off midplane fields are computed in agreement with Maxwell's equations.

In the case of all other bending elements, certain models have to be used to describe the details of the fringe–field fall off in the Enge model. In the case of the inhomogeneous magnet **MS**, the inhomogeneity of the field which is determined by the distance to the center of deflection is modulated with an Enge fall off. In the case of the combined function magnet **MC**, the inhomogeneity of the field is modulated by a fall off function following as in the case of the dipole whose edge angles and curvatures are chosen to match the linear and quadratic parts of the curves described by S1 and S2. The remaining higher order edge effects are superimposed by nonlinear kicks before and after the element.

For general purpose bending magnets, it is rather difficult to formulate field models that describe all details to a high accuracy, and hence the accuracy of the computation of aberrations is limited by these unavoidable deficiencies. In case field measurements are available, the general element **GE** allows a detailed analysis of such measured data.

3.3.5 Wigglers and Undulators

COSY INFINITY allows the computation of the maps of wigglers. For the midplane field inside the wiggler, we use the following model:

$$B_m(x,z) = B_0 \cos\left(\frac{2\pi}{\lambda} z + k \cdot z^2\right)$$

At the entrance and exit, the main field is tapered by an Enge function

$$B(x,z) = \frac{B_m(x,z)}{1 + \exp\left(a_1 + a_2 z/d + ... + a_{10}(z/d)^9\right)}$$

The wiggler is represented by the following routine:

WI < B_0 > < λ > <L> < d > <k> <I> <A> ;

where L is the length and d is the half gap. If I=0, the fringe field is modeled with some default values of the coefficients a_i. If I=1, the user is required to supply the values of a_1 to a_{10} for the entrance fringe field in the array A. The exit fringe field is assumed to have the same shape as the entrance fringe field.

3.3.6 Cavities

There is a model for a simple cavity in COSY INFINITY. It provides an energy gain that is position dependent but occurs over an infinitely thin region. The voltage of the cavity as a function of position and time is described by

$$V = P(x,y) \cdot \sin\left(2\pi(\nu \cdot t + \phi/360)\right),$$

so that ν is the frequency in Hertz, ϕ is the phase in degrees at which the reference particle enters the cavity. The peak voltage P is given in kV.

The cavity is represented by the following routine:

RF <V> <I> < ν > < ϕ > < d > ;

where V is a two dimensional array containing the coefficients of a polynomial of order I describing the influence of the position as

$$P(x,y) = \sum_{j,k=0}^{I} V(j+1, k+1) \cdot x^j \cdot y^k$$

and d is the aperture.

3.3.7 Cylindrical Electromagnetic Lenses

COSY INFINITY also allows the use of a variety of cylindrical lenses, in which focusing effects occur only due to fringe–field effects. The simplest such element consists of only one ring of radius d that carries a current I. The on-axis field of such a ring is given by

$$B(s) = \frac{\mu_0 I}{2d} \cdot \frac{1}{\left(1 + (s/d)^2\right)^{3/2}} , \qquad (1)$$

using the Biot-Savart law. This current ring is represented by the procedure

CMR $< I > < d >$;

A magnetic field of more practical significance is that of the so-called Glaser lens, which represents a good approximation of the fields generated by strong magnetic lenses with short magnetic pole pieces [28]. The lens is characterized by the field

$$B(s) = \frac{B_0}{1 + (s/d)^2} ,$$

where B_0 is the maximum field in Tesla and d is the half-width of the field. The Glaser lens is invoked by calling the procedure

CML $< B_0 > < d >$;

A third and a fourth magnetic round lenses available in COSY are solenoids.

CMSI $< I > < n > < d > < l >$;

invokes the solenoid with the theoretical field distribution

$$B(s) = \frac{\mu_0 I n}{2} \left(\frac{s}{\sqrt{s^2 + d^2}} - \frac{s - l}{\sqrt{(s - l)^2 + d^2}} \right) ,$$

obtained from (1), where I is the current, n the number of turns per meter, d the radius and l the length of the solenoid.

Another solenoid available in COSY has the the following field distribution:

$$B(s) = \frac{B_0}{2 \tanh[l/2d]} \left(\tanh[s/d] - \tanh[(s - l)/d]\right)$$

where B_0 is the field strength at the center of the solenoid, d is its aperture and l its length. The field goes down at the fringes much more quickly compared to the theoretical one for CMSI. This is invoked by the procedure

CMS $< B_0 > < d > < l >$;

There is a fifth magnetic round lens with a Gaussian potential

$$A(s) = A_0 \cdot \exp[-(s/d)^2]$$

which is invoked with the procedure

CMG $< A_0 > < d >$;

Besides the magnetic round lenses, there are various electrostatic round lenses. The element

CEL $< V_0 > < d > < L > < c >$;

lets an electrostatic lens consisting of three tubes act on the map. This lens is often called three-cube einzel lens. Figure 1 shows the geometry of the lens which consists of three coaxial tubes with identical radii d, of which the outer ones are on ground potential and the inner one is at potential V_0 in kV. The length of the middle tube is L, and the distance between the central tube and each of the outside tubes is c. Such an arrangement of three tubes can be approximated to produce an axis potential of the form

$$V(s) = -\frac{V_0}{2\omega c/d}\left(\ln\frac{\cosh\left(\omega(s+L/2)/d\right)}{\cosh\left(\omega(s+L/2+c)/d\right)} + \ln\frac{\cosh\left(\omega(s-L/2)/d\right)}{\cosh\left(\omega(s-L/2-c)/d\right)}\right),$$

where the value of the constant ω is 1.315. For details, refer to [26].

There is another electrostatic lens,

CEA $< V_0 > < d > < L > < c >$;

which lets a so-called three-aperture einzel lens act on the map. The geometry of the lens is shown in Figure 1. The outer apertures are on ground potential and the inner one is at potential V_0. The axis potential of the system can be approximated to be

$$V(s) = \frac{V_0}{\pi c}\left[(s+L/2+c)\tan^{-1}\left(\frac{s+L/2+c}{d}\right) + (s-L/2-c)\tan^{-1}\left(\frac{s-L/2-c}{d}\right)\right.$$
$$\left. -(s+L/2)\tan^{-1}\left(\frac{s+L/2}{d}\right) - (s-L/2)\tan^{-1}\left(\frac{s-L/2}{d}\right)\right].$$

An often used approximation for electrostatic lenses is described by a potential distribution of the following form

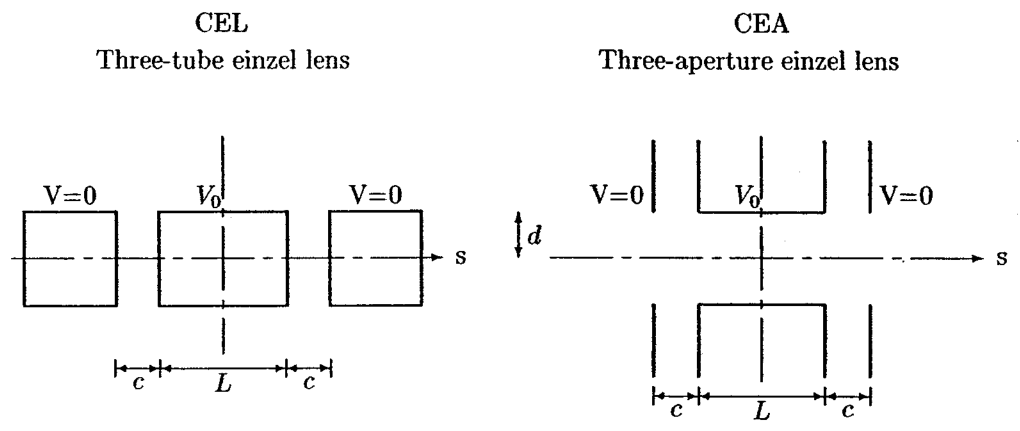


Figure 1: The constitution of electrostatic lenses of the procedure CEL and CEA

$$V(s) = V_0 \cdot \exp[-(s/d)^2] \ .$$

A lens with this field can be invoked by calling the routine

CEG $< V_0 > < d >$;

All round lenses are computed using COSY's 8th order Runge Kutta DA integrator. The computational accuracy can be changed from its default of 10^{-10} using the procedure ESET (see index).

3.3.8 General Particle Optical Elements

In this section, we present procedures that allow the computation of an arbitrary order map for a completely general optical element whose fields are described by measurements.

One way to compute a map of a general optical element is to use the procedure **GE**, which uses measurements along the independent variable s. Its use ranges from special measured fringe fields over dedicated electrostatic lenses to the computation of maps for cyclotron orbits. It can also be used to custom build new elements that are frequently used (see section 5.7 on page 55).

GE <n> <m> <S> <H> <V> <W> ;

lets an arbitrary particle optical element act on the map. The element is characterized by arrays specifying the values of multipole strengths at the n positions along the independent variable contained in the array S. The array H contains the corresponding curvatures at the positions in S. V and W contain the electric and magnetic scalar

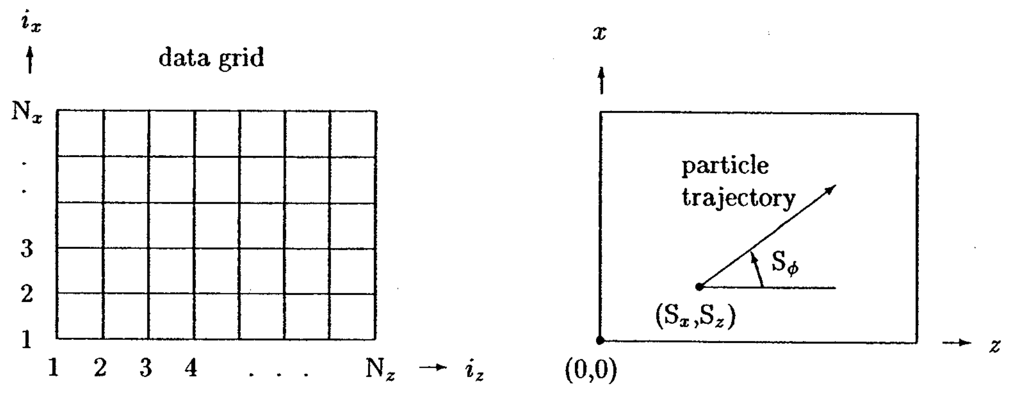


Figure 2: The specification of measured field data of the procedure MF

potentials in S.

The elements in V and W have to be DA variables containing the momentary derivatives in the x direction (variable 1) and s direction (variable 2). m is the order of the s-derivatives. One way to compute these DA variables is to write two COSY functions that compute V and W as a function of x and s. Suppose these functions are called VFUN(X,S) and WFUN(X,S), then the requested DA variable can be stored in V and W with the commands

```
V(I) := VFUN(0+DA(1),S(I)+DA(2)) ;
W(I) := WFUN(0+DA(1),S(I)+DA(2)) ;
```

Another way to compute a map of a general optical element is to use the procedures **MGE** and **MF**. While **MGE** uses measured data of the field along the independent variable s, **MF** uses measured data of the field on the midplane in cartesian coordinates [34] [35]. For deflecting elements, **MF** is more direct for users. The command

MF <s> <BY> $<N_x>$ $<N_z>$ $<\Delta x>$ $<\Delta z>$ <S> $<\boldsymbol{d}>$ $<S_x>$ $<S_z>$ $<S_\phi>$;

lets an arbitrary particle optical element act on the map. The element is characterized by a two dimensional array $\mathrm{BY}(i_x, i_z)$ specifying the values of the field strength in the y direction B_y in the midplane along an equidistant grid. Figure 2 shows how the data grid is specified and the cartesian coordinates corresponding to the data grid. N_x and N_z are the numbers of measured data grid points in the x and z direction. Δx and Δz are the lengths of each grid in the x and z direction. As shown in Figure 2, S_x and S_z are the values of (x, z) coordinates of the starting point of the reference particle in the element, and S_ϕ is the angle (degree) at the starting point of the reference particle. s is the arclength along the reference particle, and d is the aperture.

The interpolation to evaluate the values of the field strength in the element is done by the method of Gaussian interpolation. S describes the width of the Gaussian. The value of the field strength B_y at the coordinates point (x, z) is interpolated by the following equation.

$$B_y(x,z) = \sum_{i_x}\sum_{i_z} \mathrm{BY}(i_x, i_z)\frac{1}{\pi \mathrm{S}^2} \exp\left[-\frac{(x - x(i_x))^2}{\Delta x^2 \mathrm{S}^2} - \frac{(z - z(i_z))^2}{\Delta z^2 \mathrm{S}^2}\right] ,$$

where $x(i_x)$ and $z(i_z)$ are the coordinates of the (i_x, i_z)-th grid point. A note has to be made to choose the suitable S. If S is too small, the mountains structure of gaussians is observed. On the other hand, if S is too large, the original value supplied by the measured data is washed out. The suitable value of S depends on the original function shape of the measured data. For constant fields, the suitable S may be about 1.8. For quickly varying fields, it may be about 1.0. And larger values of S provide more accurate evaluation of the derivatives. In general, suitable values of S may be around $1.2 < \mathrm{S} < 1.6$.

Another note about the Gaussian interpolation is, since a gaussian function falls down quickly, the time consuming summation over all the gaussians is not necessary. The summation is well approximated by the 8S neighboring gaussians of each side. For the value outside the area, the edge value is used. When such a situation happens, the total number of such points is reported as follows:

```
*** WARNING IN MF, OUT OF RANGE OF DATA AT      123 POINTS
```

In the case of quickly varying fields, a larger area of data has to be prepared.

Since the procedure **MF** consumes the memory size in the program, a small size is prepared for the shipping of COSY.FOX . If the measured data is bigger than 20*20 gridpoints, change the size for the array in COSY.FOX in the following line.

```
VARIABLE MFD 1 20 20 ; {DATA FOR MEASURED FIELD}
```

At the same time, also in user's own program. Often the memory size of the system has to be increased. For that purpose, change the value of LVAR in the PARAMETER statements in FOXY.FOP, FOXGRAF.FOP, DAFOX.FOP .

MGE is similar to **MF** except that data for multipole terms are specified. It can be used for multipoles whose field distribution cannot be described analytically by Enge functions etc. The command

MGE $<$ NP $>$ $<$A$>$ $<$N$_s$ $>$ $< \Delta s >$ $<$S$>$ $< d >$;

lets a superimposed magnetic multipole based on measured data act on the map. N_s is the number of measured data grid points along s, where each point is spaced equidistantly by Δs. S describes the width of the Gaussian as **MF**, and d is the aperture. NP is the maxinum number of multipole components. The measured data is passed by a two dimensional array $A(i_p, i_s)$, where i_p denotes the multipole component as 1 for quadrupole, 2 for sextupole and so on, and $i_s = 1, ..., N_s$ denotes the i_s-th data point. A should be prepared to represent the field strength of the i_p-th component at the pole tip at the i_s-th position.

The same interpolation method is used as **MF**, so do the same cautions apply including the one on the memory size. The value of field strength B of the i_p-th component at the coordinates point s is interpolated as

$$B(i_p, s) = \sum_{i_s} A(i_p, i_s) \frac{1}{\sqrt{\pi} S} \exp\left[-\frac{(s - s(i_s))^2}{\Delta s^2 S^2}\right] ,$$

where $s(i_s) = \Delta s \cdot (i_s - 1)$ is the coordinate of the i_s-th grid point. Note that the total length of the element is $\Delta s \cdot (N_s - 1)$.

The map of the general element is computed using COSY's 8th order Runge Kutta DA integrator. The computational accuracy can be changed from its default of 10^{-10} using the procedure **ESET** (see index).

3.3.9 Glass Lenses and Mirrors

COSY INFINITY also allows the computation of higher order effects of general glass optical systems. At the present time, it contains elements for spherical lenses and mirrors, parabolic lenses and mirrors, and general surface lenses and mirrors, where the surface is described by a polynomial. There is also a prism. All these elements can be combined to systems like particle optical elements, including misalignments. The dispersion of the glass can be treated very elegantly by making the index of refraction a parameter using the function **PARA**.

The command

GLS <R1> <R2> <N> <L> < *d* > ;

lets a spherical glass lens act on the map. R1 and R2 are the radii of the spheres; positive radii correspond to the center of the sphere to be to the right. N is the index of refraction, L is the thickness, and d the aperture radius. The command

GL <P1> <I1> <P2> <I2> <N> <L> < *d* > ;

lets a glass lens whose surface is specified by two polynomials of orders I1 and I2 act on the map. P1 and P2 are two dimensional arrays containing the coefficients of the polynomials in x and y that describe the s position of the entrance and exit surface as a function of x and y in the following way:

$$\begin{aligned} P(x,y) &= \sum_{k,l}^{I} P(k+1,l+1)x^k y^l \\ &= P(1,1) + P(2,1)\cdot x + P(1,2)\cdot y + ... \end{aligned}$$

N is the index of refraction, L the thickness of the lens and d its aperture. The command

GP <PHI1> <PHI2> <N> <L> < d > ;

lets a glass prism act on the map. PHI1 and PHI2 are the entrance and exit angles measured with respect to the momentary reference trajectory, N is the index of refraction, L the thickness along the reference trajectory, and d is the aperture radius.

Besides the refractive glass optical elements, there are mirrors. In the following mirror elements, d is the aperture radius. The command

GMS <R> < d > ;

lets a spherical mirror with radius R act on the map. The command

GMP <R> < d > ;

lets a parabolic mirror with central radius of curvature R act on the map. The command

GMF <PHI> < d > ;

lets a flat mirror with the tilt angle PHI act on the map. The command

GM <P> <I> < d > ;

lets a general glass mirror act on the map. P is a two dimensional array containing the coefficients of the polynomial in x and y that describes the surface in the same way as with **GL**, and is the the dimension.

3.4 Lattice Converters

There are tools to convert existing lattices described in the other formats into COSY language. The following subsections explain the currently available lattice converters. The converters are web-based, and the links to the web pages of the converters can be found in http://cosy.nscl.msu.edu/.

We appreciate receiving the other converters into COSY language written by users to be available to the other users.

3.4.1 MAD Input

Many existing accelerator lattices are described in the MAD standard [21, 22]. To allow the use of such MAD lattices in COSY, there is a conversion utility that transforms MAD lattices to the COSY lattices. This utility was originally written by Roger Servranckx using the original MAD compiler source code which was written by Christopher Iselin. The current program has been adjusted to MAD version 8.22 by Weishi Wan and Kyoko Makino. The MAD to COSY converter is provided on the web at http://cosy.nscl.msu.edu/.

The converter is based on MAD version 5. The important beamline elements are translated into the respective ones in COSY; these include drifts, multipoles, superimposed multipoles, and bends. Some elements supported by MAD are translated to drifts and may have to be adjusted manually.

To generate a COSY deck from a MAD deck, the end of the MAD deck should have the form

```
USE, <name of beamline>
     COSY
     STOP
```

where according to the MAD syntax, the USE command specifies the beamline to be translated, and the command COSY actually generates the COSY source.

3.4.2 SXF Input

The SXF format (Standard eXchange Format) is meant to be a general lattice description language and is intended to facilate the cooperation between different groups and the comparison of results obtained with different codes. The language specifications that are in most parts very similar to MAD were developed by H.Grote, J.Holt, N.Malitsky, F.Pilat, R.Talman, G.Tahern and W.Wan .

The SXF to COSY converter is provided on the web at http://cosy.nscl.msu.edu/.

3.5 Misalignments

The differential algebraic concept allows a particularly simple and systematic treatment of misalignment errors in optical systems. Such an error is represented by a coordinate change similar to the one discussed in section 4.1. COSY offers three different misalignment commands. The first command

SA <DX> <DY> ;

offsets the optic axis by DX in x direction and DY in y direction. DX and DY are counted positive if the optic axis is shifted in direction of positive x and y, respectively. The command

TA <AX> <AY> ;

represents a tilt of the optic axis by an angle in degrees of AX in x direction and AY in y direction. AX and AY are counted positive if the direction of tilt is in the direction of positive x and y, respectively. The command

RA <ANGLE > ;

represents a rotation of the optic axis around ANGLE measured in degrees. ANGLE is counted positive if the rotation is counterclockwise if viewed in the direction of the beam. The routine **RA** can be used to rotate a given particle optical element by placing it between counteracting rotations. This can for example be used for the study of skew multipoles. However, note that it is not possible to rotate different multipole components by different angles. This can be achieved with the routines **MMS** and **EMS** discussed in section 3.3.1.

In order to simulate a single particle optical element that is offset in positive x direction, it is necessary to have the element preceeded by an axis shift with negative value and followed by an axis shift with positive value. Similarly simple geometric considerations tell how to treat single tilted and rotated elements.

The misalignment routines can also be used to study beams that are injected off the optical axis of the system. In this case, just one of each misalignment commands is necessary at the beginning of the system.

We note that the misalignment routines, like most other COSY routines, can be called both with real number and differential algebraic arguments, in particular using the **PARA** argument (see section 5.2). The first case allows the simulation of a fixed given misalignment, whereas the second case allows to compute the map depending on the misalignment.

In the first case, the values of the computed transfer map are only approximate if **SA** and **TA** are used. The accuracy increases with decreasing misalignments and increasing calculation orders. For the study of misalignments of elements, the actual accuracy is usually rather high since the values of the misalignments are usually very small. In the case of a deliberate offset of the beam, for example for the study of injection and extraction processes, it may be necessary to increase the computation order to obtain accurate results. In the second case, the results are always accurate. The command **RA** always produces accurate results in both cases.

4 Analyzing Systems with COSY

4.1 Image Aberrations

Very often not the matrix elements of the transfer map are of primary significance, but rather the maximum size of the resulting aberration for the phase space defined with **SB** and the parameters defined with **SP**. COSY provides two tools to obtain the aberrations directly. The command

PA <unit> ;

prints all aberrations to unit in a similar way as **PM**. If not all aberrations are of interest, the COSY function

MA (<phase space variable>,<element identifier>)

returns the momentary value of the aberration. The phase space variable is a number from 1 to 6 corresponding to x, a, y, b, t, d, and the element identifier is an integer whose digits denote the above variables. For example, **MA**(1,122) returns the momentary value of the aberration due to the matrix element (x, xaa).

For comparison and other reasons, it is often helpful to express the map in other coordinates than those used by COSY (see section 3.2.1, for example the ones used in TRANSPORT [3] and GIOS [5, 19]. The routine

PT <unit> ;

prints the map in Transport and GIOS coordinates to unit.

We want to point out that in the differential algebraic concept, it is particularly simple to perform such nonlinear coordinate changes to arbitrary orders. In order to print maps in yet different coordinates, the user can make a procedure that begins with a unity map, applies the transformation to COSY coordinates, applies the COSY map, and then applies the transformation back to the original coordinates.

4.2 Analysis of Spectrographs

To first order, the resolution $\Delta\delta$ of an imaging spectrograph is given by the following simple formula:

$$\Delta\delta = \frac{(x,x)\cdot 2X_0}{(x,d)}\cdot$$

where X_0 is the half width of the slit or aperture at the entrance of the device. Here δ can be any one of the quantities δ_k, δ_m and δ_z, and it is assumed that to first order, the final position does not depend on the other quantities, or all particles have the same initial values for the other quantities.

In all but the simplest spectrographs, however, it is important to consider higher order effects as well as the finite resolution of the detectors. Usually these effects decrease the resolution, more so for larger initial phase spaces and low detector resolutions. The resolution of the spectrograph under these limitations can be computed with the following command

AR <MAP> <X> <A> <Y> <B> <D> <PR> <N> <R> ;

where MAP is the map of the spectrograph to be studied, X, A, Y, B and D are the half widths of the beam at the entrance of the spectrograph, PR is the resolution of the detector, and R is the resulting resolution of the spectrograph. To compute the resolution, a total of N particles are distributed randomly and uniformly within a square initial phase space and then sent through the map. Then the measurement error is introduced by adding a uniformly distributed random number between -PR and PR to the x coordinate. The width of the resulting blob of measurements is computed, where it is assumed that the blob is again filled uniformly.

In many cases the resolution of spectrographs can be increased substantially with the technique of trajectory reconstruction [36]. For this purpose, positions of each particle are actually measured in two detector planes, which is equivalent to knowing the particle's positions and directions.

Assuming that the particle went through the origin, the energy of the particle is uniquely determined by some complicated nonlinear implicit equations [37]. Using DA methods, it is possible to solve these equations analytically and relate the energy of the particle to the four measured quantities. Besides the energy, it is also possible to compute the initial angle in the dispersive plane, the initial position in the non-dispersive plane, and the angle in the non-dispersive plane. The accuracy of these equations is limited only by the measurement accuracy and by the entering spot size in the dispersive plane. This is performed by the command

RR <MAP> <X> <A> <Y> <B> <D> <PR> <AR> <N> <O> <MR> <R> ;

where the parameters are as before, except that AR is the resolution in the measurement of the angle, and O is the order to which the trajectory reconstruction is to be performed. On return, MR is the nonlinear four by four map relating initial a, y, b and d to the measured final x, a, y, b. Using these relationships as well as the measurement errors and the finite dispersive spot size, the resolution array R containing the resolutions of the initial a, y, b and d is computed by testing N randomly selected rays and subjecting them to statistical measurement errors similarly as with the computation of the uncorrected resolution.

4.3 Analysis of Rings

Instead of by their transfer matrices, the linear motion in particle optical systems is often described by the tune and twiss parameters. These quantities being particularly important for repetitive systems, they allow a direct answer to questions of linear stability, beam envelopes, etc. In many practical problems, their dependence on parameters is very important. For example, the dependence of the tune on energy, the chromaticity, is a very crucial quantity for the design of systems. Using the maps with knobs, they can be computed totally automatically without any extra effort. The command

TP <MU> ;

computes the tunes which are stored in the one dimensional array with three entries MU which is defined by the user. In most cases, an allocation length of 100 should be sufficient, and so the declaration of MU could read

VARIABLE MU 100 3 ;

If the system is run with parameters, MU will contain DA vectors describing how the respective tunes depend on the parameters. Note that COSY INFINITY can also compute amplitude dependent tune shifts in the framework of normal form theory. This is described in detail in this section.

For the computation of amplitude tune shifts and other characteristics of the repetitive motion, COSY INFINITY contains an implementation of the DA normal form algorithm described in [38]. This replaces the COSY implementation of the somewhat less efficient and less general mixed DA-Lie normal form [15]. Normal Form algorithms provide nonlinear transformations to new coordinates in which the motion is simpler. They allow the determination of pseudo invariants of the system, and they are the only tool so far to compute amplitude tune shifts. As pointed out in [39], chromaticities and parameter dependent tune shifts alone can be computed more directly using the command **TP** described above. The command

NF <EPS> <MA> ;

computes the normal form transformation map MA of the momentary transfer map. This variable has to be allocated by the user, and in most cases

VARIABLE MA 1000 8 ;

should be sufficient. Since the normal form algorithm has a small denominator problem, it is not always possible to perform a transformation to coordinates in which the motion is given by circles. The variable EPS sets the minimum size of a resonance denominator that is not removed. The command

TS <MU> ;

employs the normal form algorithm to compute all the tune shifts of the system, both the ones depending on amplitude and the ones depending on parameters like chromaticities, which alone can be computed more efficiently as shown above. MU is a one dimensional array with three entries which is defined by the user in a similar way to TP. On return, MU will contain the tune shifts with amplitudes and parameters as DA vectors. If the system is run with parameters, MU will contain DA vectors describing how the respective tunes depend on the amplitudes (first, third and possibly fifth exponents for x, y and t) and parameters (beginning in columns five or 7).

Note that in some cases when the system is on or very near a resonance or is even unstable, the normal form algorithm fails because of a small denominator problem. In this case, the respective tunes will be returned as zero. This also happens sometimes if the map is supposed to be symplectic yet is slightly off because of computational inaccuracies. In this case, the use of the procedure **SY** (see index) is recommended.

The normal form method can also be used to compute resonance strengths, which tell how sensitive a system is to certain resonances. Often the behavior of repetitive systems can be substantially improved by reducing the resonance strengths. These are computed with the procedure

RS <RES> ;

where upon return RES is a complex DA vector that contains the resonance strengths. The $2 \cdot N$ exponents n_i^+, n_i^- in each component describe the resonance of the tunes ν as

$$(\vec{n}_i^+ - \vec{n}_i^-) \cdot \vec{\nu}.$$

The linear and nonlinear momentum compaction $(dl/dp) \cdot p/l$ can be computed with the routine

MCM <M> <L> <C> ;

Alternatively, it also possible to compute the Energy compaction (dr_5/dr_6) with the routine

ECM <M> <L> <C> ;

Finally it is also possible to analyze the spin motion with normal form methods. The command

TSP <MU> $< \bar{n} >$ <KEY> ;

computes the parameter dependence of spin tune and the invariant spin axis $\bar{n}$. The command

TSS <MU> $< \bar{n} >$ <KEY> ;

computes the parameter and amplitude dependence of spin tune as well as the invariant spin axis $\bar{n}$. The spin tunes are stored in the one dimensional array with three entries

MU which is defined by the user, in a similar way as the array used by TP and TS. If KEY is 0, the original orbital variables are used. If KEY is not 0, the orbital variables are transformed to the parameter dependent fixed point.

Estimates on the long term stability of particle motion in a storage ring can be computed by the method of Pseudo Invariant Estimation (PIE) with

STURNS <NTU> <MAP> <E> <F> <NE> <HD> <HT> <DS> ;

These estimates are sometimes called Nekhoroshev–type estimates. The acceptances are stored in the vector E. Assuming a particle starts at emittances E/F. Then the result NTU gives a lower bound on the number of turns this particle can be mapped through MAP without getting lost. NE describes the number of points which are used to cover an ellipse when evaluating the pseudo invariants. HD describes how often the phase space between E and E/F should be divided and HT indicates to how many applications of MAP the PIE method should be applied. DS has to be chosen 0 if the quality of the pseudo invariants is to be computed by scanning with real numbers. Otherwise the phase space is covered with intervals and the density of intervals is 1/DS. Therefore DS=1 leads to rigorous bounds on the survival time. For further details please refer to the reference [40] [41].

4.4 Repetitive Tracking

COSY allows efficient repetitive tracking of particles through maps. The command

TR <N> <NP> <ID1> <ID2> <D1> <D2> <TY> <NF> <IU>;

tracks the momentary particles selected with **SR** or **ER** through the momentary map for the required number of iterations N. After each Np iterations the position of the phase space projection ID1-ID2 is drawn to unit IU. The phase space numbers 1 through 6 correspond to x, a, y, b, d, t, and the numbers -1, -2, -3 correspond to the x, y and z components of the spin. If any of these components get larger than D1, D2, they will not be drawn.

If TY is zero, the tracking is performed without symplectification. If TY is nonzero, symplectic tracking using the generating function of type |TY| (see page 46) is performed. For positive TY, the whole map is symplectified; in case TY is negative, symplectic tracking is performed by symplectifying the linear map $\mathcal{M}_L$ and representing the map $\mathcal{N} = \mathcal{M} \circ \mathcal{M}_L^{-1}$ by the generating function of type |TY|. Because the linear part of $\mathcal{N}$ is the unity map, TY can only be -2 or -3 for that purpose. If NF is zero, the points will be displayed in conventional variables, and if NF is one, they will be displayed in normal form variables.

As discussed in subsection 7.2.2, if needed the coordinates can also be output directly for future manipulation.

The algorithm used for tracking is highly optimized for speed. Using the vector data type for particle coordinates, it works most efficiently if many particles are tracked simultaneously. On scalar machines, optimum efficiency is obtained when more than about 20 particles are tracked simultaneously. On vector machines, the algorithm vectorizes completely, and for best efficiency, the number of particles should be a multiple of the length of the hardware vector.

In both cases, logistics overhead necessary for the bookkeeping is almost completely negligible, and the computation time is almost entirely spent on arithmetic. It is also worth mentioning that using an optimal tree transversal algorithm, zero terms occurring in a map do not contribute to computation time.

4.5 Symplectic Representations

In this section, we will present two different representations for symplectic maps, each one of which has certain advantages. Particle optical systems described by Hamiltonian motion satisfy the symplectic condition

$$M \cdot J \cdot M^t = J$$

where M is the Jacobian Matrix of partial derivatives of $\mathcal{M}$, and J has the form

$$J = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix}$$

As long as there is no damping, all particle optical systems are Hamiltonian, and so the maps are symplectic up to possibly computation errors if they are generated numerically. There is a COSY function that determines the symplectic error of a map:

SE (<M>)

Here M is an array of DA quantities describing the map. Note that the momentary value of the transfer map is stored in the global COSY variable MAP. The value of the function is the weighted maximum norm of the matrix $(M \cdot J \cdot M^t - J)$. The weighting is done such that the maximum error on a cubic phase space with half edge W is computed. The default value for W is .1, which may be too large for many cases. The value of W can be set with the procedure

WSET <W> ;

While the orbital part of maps usually satisfies the symplectic symmetry, the spin matrix must satisfy orthogonality. Similar to the function **SE**,

OE (<SM>)

determines the orthogonality error of the spin matrix SM. The current system spin matrix is stored in the array SPNR.

In some instances, it may be desirable to symplectify maps that are not fully symplectic. While the standard elements of COSY are symplectic to close to machine precision, the low accuracy fringe–field modes (see section 3.3.4) violate symplecticity noticeably. Depending on the coarseness of the measured field data, this may also occur in the general element discussed in section 3.3.8. To a much lesser extent symplecticity is violated by intrinsic elements requiring numerical integration, like the high-precision fringe fields and the round lenses discussed in section 3.3.7. The command

SY <M> ;

symplectifies the map M using the generating function (see below) which is most accurate for the given map.

Symplectic maps can be represented by at least one of four generating functions in mixed variables:

$$F_1(q_i, q_f) \text{ satisfying } (\vec{p_i}, \vec{p_f}) = (\vec{\nabla}_{q_i} F_1, -\vec{\nabla}_{q_f} F_1)$$

$$F_2(q_i, p_f) \text{ satisfying } (\vec{p_i}, \vec{q_f}) = (\vec{\nabla}_{q_i} F_2, \vec{\nabla}_{P_f} F_2)$$

$$F_3(p_i, q_f) \text{ satisfying } (\vec{q_i}, \vec{p_f}) = (-\vec{\nabla}_{p_i} F_3, -\vec{\nabla}_{q_f} F_3)$$

$$F_4(p_i, p_f) \text{ satisfying } (\vec{q_i}, \vec{q_f}) = (-\vec{\nabla}_{p_i} F_4, \vec{\nabla}_{p_f} F_4)$$

In the generating function representation there are no interrelationships between the coefficients due to symplecticity like in the transfer map, so the generating function representation is more compact. Furthermore, it is often an important tool for the symplectification of tracking data. The command

MGF <M> <F> <I> <IER> ;

attempts to compute the I th generating function of the specified map M. If IER is equal to zero, this generating function exists and is contained in F. If IER is nonzero, it does not exist. While in principle, any generating function that exists represents the map, especially for high order maps, certain inaccuracies often result for numerical reasons. If I is chosen to be -1, the generating function representing the linear part of the map best is determined. For I equal to -2, the generating function representing the whole map best is computed. The case I $= -2$ is very expensive computationally and should only be used in crucial cases for high orders. In both cases, on return I contains the number of the chosen generating function.

The map which corresponds to a generating function F of type I is obtained by

GFM <M> <F> <I> ;

Other redundancy free representations of symplectic transfer maps are Lie factorizations including the Dragt-Finn factorization [42, 43, 8] . They are based on Lie transformation operators of the form

$$\exp(:f:) = 1 + :f: + \frac{:f:^2}{2} + \ldots$$

where f is a function of the canonical coordinates q_i and p_i. The colon denotes a Poisson bracket waiting to happen, i.e. $:f:g = \{f,g\}$. When $\vec{x}_f$ describes a final set of canonical coordinates with $\vec{x} = (q_1, p_1, \ldots, q_n, p_n)$ and $\vec{x}_i$ describes an initial set, then $\vec{x}_f = \exp(:f:)\vec{x}_i$ is a symplectic mapping. Those Lie transformation operators have the property

$$e^{:f:}(g(\vec{x})) = g(e^{:f:}\vec{x})$$

for any function $g : \Re^{2n} \rightarrow \Re$ with n being the dimension of the required configuration space. Therefore we find

$$e^{:f:}(e^{:g:}\vec{x}) = (e^{:g:}\vec{x}) \circ (e^{:f:}\vec{x})$$

The circle $\circ$ symbolizes the composition of maps. Two composed symplectic maps are therefore represented by the product of their Lie transformation operators in reversed order. As an example, a symplectic map can be written in the form

$$\tilde{L}e^{:f_>:}\vec{x} + \vec{C}$$

were $\tilde{L}$ is an operator such that $\tilde{L}\vec{x}$ is the linear part of the map, $f_>$ is a polynomial in the x_i containing only orders higher than 2. Finally $\vec{C}$ represents the constant part of the map. As mentioned previously this representation is equal to

$$(e^{:f_>:}\vec{x}) \circ (\tilde{L}\vec{x}) + \vec{C}$$

Besides this factorization, there are various others that are similar and have certain advantages [8]. They are shown in the table below. As shown in [8], it is one of the strong points of the map representation and the differential algebraic techniques that the computation of these Dragt-Finn factorization is possible to arbitrary order with a relatively simple algorithm. It is actually much easier to compute them from the map than using Lie algebraic techniques alone. The command

MLF <MA> <C> <M> <F> <I> ;

computes the factorization from the transfer map MA. On return, the vector C contains the constant part, M the linear part and F contains the f_i from the table. In case of the last four factorization F has to be an array. I is the identifier of the factorization following the numbering in the table.

$$
\begin{array}{rcl}
1 & : & \mathcal{M}(\vec{x}) =_n \tilde{L}\exp(:f_>:)\vec{x} + \vec{C} \\
-1 & : & \mathcal{M}(\vec{x}) =_n \exp(:f_>:)\tilde{L}\vec{x} + \vec{C} \\
2 & : & \mathcal{M}(\vec{x}) =_n \tilde{L}\exp(:f_3:)\exp(:f_4:)\ldots\exp(:f_{n+1}:)\vec{x} + \vec{C} \\
-2 & : & \mathcal{M}(\vec{x}) =_n \exp(:f_{n+1}:)\ldots\exp(:f_3:)\tilde{L}\vec{x} + \vec{C} \\
3 & : & \mathcal{M}(\vec{x}) =_{2^{n+1}} \tilde{L}\exp(:f_{3,3}:)\exp(:f_{4,5}:)\exp(:f_{6,9}:)\ldots\exp(:f_{(2^n+2),(2^{n+1}+1)}:)\vec{x} + \vec{C} \\
-3 & : & \mathcal{M}(\vec{x}) =_{2^{n+1}} \exp(:f_{2^n+2,2^{n+1}+1}:)\ldots\exp(:f_{6,9}:)\exp(:f_{4,5}:)\exp(:f_{3,3}:)\tilde{L}\vec{x} + \vec{C}
\end{array}
$$

Here f_i denotes homogeneous polynomials of exact order i and $f_{i,j}$ polynomials with orders from i to j. Given a factorization, the command

LFM <MA> <C> <M> <F> <I> ;

calculates the according map. The command

LFLF <C> <M> <F> <P> <I> <J> ;

computes the factorization of type J with exponent P from a factorization of type I with exponent F. Without the map representation this would be a very elaborate task, because the Campbell-Baker-Hausdorff formula would be needed to the appropriate order.

5 Examples

This section provides several examples for the use of core features of COSY. The code DEMO.FOX which is sent out with COSY contains many more programs that can serve as demonstrations. Further ideas how to use the COSY language can also be obtained by studying COSY.FOX.

5.1 A Simple Sequence of Elements

After having discussed the particle optical elements and features available in COSY INFINITY in the previous sections, we now discuss the computation of maps of simple systems.

We begin with the computation of the transfer map of a quadrupole doublet to tenth order. Here the COSY input resembles the input of many other optics codes [6, 5].

```
INCLUDE 'COSY' ;
```

```
PROCEDURE RUN ;
   OV 10 2 0 ;       {order 10, phase space dim 2, # of parameters 0}
   RP 10 4 2 ;       {kinetic energy 10 MeV, mass 4 amu, charge 2}
   UM ;              {sets map to unity}
   DL .1 ;           {drift of length .1 m}
   MQ .2 .1 .05 ;    {quad; length .2 m, field .1 T, aperture .05 m}
   DL .1 ;
   MQ .2 -.1 .05 ;   {defocussing quad}
   DL .1 ;
   PM 11 ;           {prints map to unit 11}
   ENDPROCEDURE ;
RUN ; END ;
```

The first few lines of the resulting transfer map on unit 11 look like this:

```
  0.7084974     -0.1798230      0.0000000E+00 0.0000000E+00 0.0000000E+00 100000
  0.6952214       1.234984      0.0000000E+00 0.0000000E+00 0.0000000E+00 010000
  0.0000000E+00 0.0000000E+00  1.234984     -0.1798230      0.0000000E+00 001000
  0.0000000E+00 0.0000000E+00 0.6952214      0.7084974      0.0000000E+00 000100
 -0.7552782E-01-0.5173663E-01 0.0000000E+00 0.0000000E+00 0.0000000E+00 300000
  0.2751172      0.1728297      0.0000000E+00 0.0000000E+00 0.0000000E+00 210000
 -0.4105719     -0.2057598      0.0000000E+00 0.0000000E+00 0.0000000E+00 120000
  0.3541071      0.8137949E-01 0.0000000E+00 0.0000000E+00 0.0000000E+00 030000
  0.0000000E+00 0.0000000E+00 0.5676311E-01-0.5150457E-01 0.0000000E+00 201000
```

The different columns correspond to the final coordinates x, a, y, b and t. The lines contain the various expansion coefficients, which are identified by the exponents of the initial condition. For example, the last entry in the third column is the expansion coefficient (y, xxy).

5.2 Maps with Knobs

The DA approach easily allows to compute maps not only depending on phase space variables, but also on system parameters. This can be very helpful for different reasons. For example, it directly tells how sensitive the system is to errors in a particular quantity. In the same way it can be used to find out ideal positions to place correcting elements. Furthermore, it can be very helpful for the optimization of systems, and sometimes very fast convergence can be achieved with it (for details, see section 7.1).

In the context of COSY INFINITY, the treatment of such system parameters or knobs is particularly elegant.

In the following example, we compute the map of a system depending on the strength of one quadrupole. The COSY function PARA(I) is used, which identifies the quantity as parameter number I by turning it into an appropriate DA vector.

```
INCLUDE 'COSY' ;
PROCEDURE RUN ;
   OV 5 2 1 ;                  {order 5, phase space dim 2, parameters 1}
   RP 10 4 2 ;                 {sets kinetic energy, mass and charge}
   UM ;
   DL .1 ;
   MQ .2 .1*PARA(1) .05 ;      {quadrupole; now field is a DA quantity}
   DL .1 ;
   MQ .2 -.1 .05 ;
   DL .1 ;
   PM 11 ;                     {prints map depending on quad strength}
   ENDPROCEDURE ;
RUN ; END ;
```

In this context it is important that the COSY language supports freedom of types at compile time; so the second argument of the quad can be either real or DA. For details, consult section 6.

The idea of maps with knobs can also be used to compute the dependence on the particle mass and charge as well as on energy in case time of flight terms are not needed. In the following example, the map of the quad doublet is computed including the dependence on energy, mass and charge.

```
INCLUDE 'COSY' ;
PROCEDURE RUN ;
   OV 5 2 3 ;                  {order 5, phase space dim 2, parameters 3}
   RP 10*PARA(1) 4*PARA(2) 2*PARA(3) ;    {sets kinetic energy, mass
                                           and charge as DA quantities}
   UM ;
   DL .1 ;
   MQ .2 .1 .05 ;
   DL .1 ;
   MQ .2 -.1 .05 ;
   DL .1 ;
   PM 11 ;                     {prints map with dependence on energy,
                                mass and charge, to unit 11}
   ENDPROCEDURE ;
RUN ; END ;
```

5.3 Grouping of Elements

Usually it is necessary to group a set of elements together into a cell. For example, since most circular accelerators are built of several at least almost identical cells, it is desirable to refer to the cell as a block. Similar situations often occur for spectrometers or microscopes if similar quad multiplets are used repetitively.

Grouping is easily accomplished in COSY by just putting the elements into a procedure. In the following example, the strength of a quadrupole in the cell of an accelerator is adjusted manually such that the motion in both planes is stable. Since the motions are stable if the two traces are less than two in magnitude, the map is printed to the screen which allows a direct check.

```
INCLUDE 'COSY' ;
PROCEDURE RUN ; VARIABLE QS 1 ;     {declare a real variable}
   PROCEDURE CELL Q H1 H2 ;         {defines a cell of a ring}
      DL .3 ; DI 10 20 .1 0 0 0 0 ; DL .1 ; MH .1 H1 .05 ;
      DL .1 ; MQ .1 Q .05 ;     DL .3 ; MH .1 H2 .05 ;
      ENDPROCEDURE ;
   OV 3 2 0  ; RPP 1000 ;     {third order, one GeV protons}
   QS := .1 ;                   {set initial value for quad}
   WHILE QS#0 ; WRITE 6 ' GIVE QS ' ; READ 5 QS ;
      UM ; CELL QS 0 0 ;  PM 6 ; WRITE 6 ME(3,3) ;
      ENDWHILE ;
   ENDPROCEDURE ; RUN ; END ;
```

Obviously, such groupings can be nested if necessary, and parameters on which the elements in the group depend can be passed freely. Note that calling a group entails that all elements in it are executed; so grouping is not a means to reduce execution time, but a way to organize complicated systems into easily manageable parts. Reduction of execution time can be achieved by saving maps of subsystems that do not change using SM and AM discussed above.

5.4 Optimization

One of the most important tasks in the design of optical systems is the optimization of certain parameters of the system to meet certain specifications. Because of the importance of optimization, there is direct support from the COSY language via the **FIT** and **ENDFIT** commands. COSY provides several FORTRAN based optimizers; a detailed description of the optimizers available in COSY can be found in section 7.1.

In the first example we illustrate a simple optimization task: to fit the strengths of the quadrupoles of a symmetric triplet to perform stigmatic point-to-point imaging. To monitor the optimization process, the momentary values of the quad strengths and the objective function are printed to the screen. Furthermore, a graphic display of the system at each step of the optimizer is displayed in two graphic windows, here identified with units -101 and -102, one for each phase space projection, creating a movie-like effect.

Subsection 7.2.2 beginning on page 70 lists the graphics drivers currently supported in COSY. At the end, the final pictures of the x and y projection of the system is printed in LaTeX picture format, identified with unit -7, for inclusion in this manual.

```
INCLUDE 'COSY' ;
PROCEDURE RUN ;
   VARIABLE Q1 1 ; VARIABLE Q2 1 ; VARIABLE OBJ 1 ;
   PROCEDURE TRIPLET A B ;
      MQ .1 A .05 ; DL .05 ; MQ .1 -B .05 ; DL .05 ; MQ .1 A .05 ;
      ENDPROCEDURE ;
   OV 1 2 0 ;
   RP 1 1 1 ;
   SB .15 .15 0 .15 .15 0 0 0 0 0 0 ;
   Q1 := .5 ; Q2 := .5 ;
   FIT Q1 Q2 ;
      UM ; CR ; ER 1 4 1 4  1 1 1 1 ;
      BP ; DL .2 ; TRIPLET Q1 Q2 ; DL .2 ; EP ;
      PP -101 0 0 ; PP -102 0 90 ;
      OBJ := ABS(ME(1,2))+ABS(ME(3,4)) ;
      WRITE 6 'STRENGTHS Q1, Q2, OBJECTIVE FUNCTION: ' Q1 Q2 OBJ ;
      ENDFIT 1E-5 1000 1 OBJ ; PP -7 0 0 ; PP -7 0 90 ;
   ENDPROCEDURE ; RUN ; END ;
```

The x projection picture of the system after optimization is shown in Figure 3. The LaTeX file of this picture, lpic001.tex, produced by COSY has been copied into the LaTeX source of this manual.

Besides providing "canned" optimization strategies, the COSY language allows to follow one's own path of optimizing a system, which typically consists of several runs with varying parameters and subsequent optimizations.

In the following example, the goal is to vary several parameters of the system manually, fit the quad strengths, and then look at the spherical aberrations. This process is repeated by inputting different values for the parameters until the spherical aberrations have been reduced to a satisfactory level. When this is achieved, the pictures of the system is output directly to PostScript files, identified with unit -10, with the names pic001.ps and pic002.ps.

```
INCLUDE 'COSY' ;
PROCEDURE RUN ;
   VARIABLE Q1 1 ; VARIABLE Q2 1 ; VARIABLE L1 1 ; VARIABLE L2 1 ;
   VARIABLE OBJ 1 ; VARIABLE ISTOP 1 ;
   PROCEDURE TRIPLET ;
      UM ; CR ; ER 1 4 1 4 1 1 1 1 ; BP ;
```

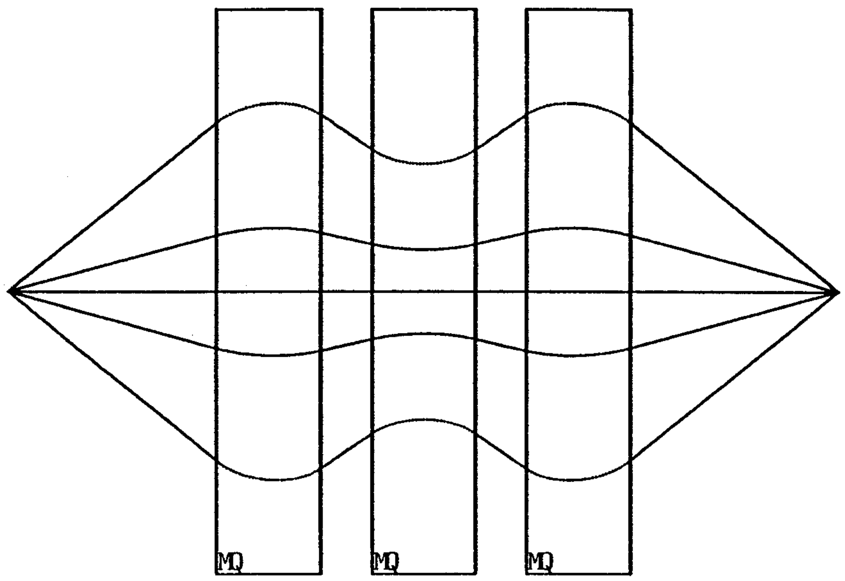


Figure 3: COSY LaTeX picture of the stigmatically focusing system

```
   DL L1 ; MQ .1 Q1 .05 ; DL L2 ; MQ .1 -Q2 .05 ;
   DL L2 ; MQ .1 Q1 .05 ; DL L1 ; EP ; PP -101 0 0 ;
   ENDPROCEDURE ;
OV 3 2 0 ; RP 1 1 1 ; SB .08 .08 0 .08 .08 0 0 0 0 0 0 ; ISTOP := 1 ;
WHILE ISTOP#0 ;
   WRITE 6 ' GIVE VALUES FOR L1, L2: ' ; READ 5 L1 ; READ 5 L2 ;
   Q1 := .5 ; Q2 := .5 ; CO 1 ;
   FIT Q1 Q2 ; TRIPLET ; OBJ := ABS(ME(1,2))+ABS(ME(3,4)) ;
      ENDFIT 1E-5 1000 1 OBJ ;
      CO 3 ; TRIPLET ;
      WRITE 6 ' SPHERICAL ABERRATION FOR THIS SYSTEM: ' ME(1,222) ;
      WRITE 6 ' CONTINUE SEARCH? (1/0) ' ; READ 5 ISTOP ;
   ENDWHILE ; PP -10 0 0 ; PP -10 0 90 ;
ENDPROCEDURE ; RUN ; END ;
```

This example shows how it is possible to phrase more complicated interactive optimization tasks in the COSY language. One can even go far beyond the level of sophistication displayed here; by nesting sufficiently many WHILE, IF and LOOP statements, it is often possible to optimize a whole system in one interactive session without ever leaving COSY. For example, the first order design in [44] which is subject to quite a number of constraints and requires a sophisticated combination of trial and optimization was performed in this way.

5.5 Normal Form, Tune Shifts and Twiss Parameters

The following example shows the use of normal form methods and parameter dependent Twiss parameters for the analysis of a repetitive system. For the sake of simplicity, we choose here a simple FODO cell that is described by the procedure CELL. The map of the cell is computed to fifth order, with the energy as a parameter. In the cell itself, the quadrupole strength is another parameter.

As a first step, the parameter dependent tunes are computed and written to unit 7, following the algorithm in [39]. Next follow the tunes depending on parameters and amplitude; this is done with DA normal form theory [38]. Finally, several other quantities and their parameter dependence are computed using the procedure GT. They include the parameter dependent fixed point, the parameter dependent Twiss parameters, as well as the parameter dependent damping (which here is unity because no radiation effects are taken into account).

```
INCLUDE 'COSY' ;
PROCEDURE RUN  ;
   VARIABLE A 100 2 ; VARIABLE B 100 2 ; VARIABLE G 100 2 ;
   VARIABLE R 100 2 ; VARIABLE MU 100 2 ; VARIABLE F 100 6 ;
   PROCEDURE CELL ;
      DL .1 ; DI 1 45 .1 0 0 0 0 ; DL .1 ; MQ .1 -.1*PARA(2) .1 ; DL .2 ;
      ENDPROCEDURE ;
   OV 5 2 2 ; RP 1*PARA(1) 1 1 ; UM ; CELL ;
   TP MU ; WRITE 7 ' DELTA DEPENDENT TUNES '          MU(1) MU(2) ;
   TS MU ; WRITE 7 ' DELTA AND EPS DEPENDENT TUNES ' MU(1) MU(2) ;
   GT MAP F MU A B G R  ;
   WRITE 7 ' DELTA DEPENDENT FIXED POINT ' F(1) F(2) F(3) F(4) ;
   WRITE 7 ' DELTA DEPENDENT ALPHAS '      A(1) A(2) ;
   WRITE 7 ' DELTA DEPENDENT BETAS '       B(1) B(2) ;
   WRITE 7 ' DELTA DEPENDENT GAMMAS '      G(1) G(2) ;
   WRITE 7 ' DELTA DEPENDENT DAMPINGS '    R(1) R(2) ;
   ENDPROCEDURE ; RUN ; END ;
```

5.6 Repetitive Tracking

In the following example, we want to study the nonlinear behaviour of a ring by a qualitative analysis of tracking data. The ring consists of 18 identical cells. Nine of these cells are packed into a half cell by the procedure HALFCELL. At execution, the system asks for the values of the strengths of the two hexapoles which influence its degree of nonlinearity. The tracking data for each setting are displayed and then also output in LaTeX format for inclusion in this manual, shown in Figure 4. In order to keep the size of the LaTeX source file limited, only 100 turns were tracked for five particles.

```
INCLUDE 'COSY' ;
PROCEDURE RUN ; VARIABLE QS 1 ; VARIABLE H1 1 ; VARIABLE H2 1 ; VARIABLE N 1 ;
   PROCEDURE CELL Q H1 H2 ;          {defines a cell of a ring}
      DL .3 ; DI 10 20 .1 0 0 0 0 ; DL .1 ; MH .1 H1 .05 ;
      DL .1 ; MQ .1 Q .05 ;     DL .3 ; MH .1 H2 .05 ;
      ENDPROCEDURE ;
   PROCEDURE HALFRING Q H1 H2 ; VARIABLE I 1 ;
      LOOP I 1 9 ; CELL Q H1 H2 ; ENDLOOP ; ENDPROCEDURE ;
   OV 3 2 0  ; RPP 1000 ;     {third order, one GeV protons}
   QS := -.05 ; H1 := .01 ;
   WHILE H1#0 ; WRITE 6 ' GIVE HEXAPOLE STRENGTHS ' ; READ 5 H1 ; READ 5 H2 ;
      UM ; HALFRING QS H1 H2 ;
      WRITE 6 ' GIVE NUMBER OF TURNS ' ; READ 5 N ;
      SR .005 0 .005 0 0 0 0 0 1 ;
      SR .01  0 .01  0 0 0 0 0 1 ;
      SR .015 0 .015 0 0 0 0 0 1 ;
      SR .02  0 .02  0 0 0 0 0 1 ;
      TR N 1  1 2  .03 .002   0 0 -101  ; CR ;
      SR .005 0 .005 0 0 0 0 0 1 ;
      SR .01  0 .01  0 0 0 0 0 1 ;
      SR .015 0 .015 0 0 0 0 0 1 ;
      SR .02  0 .02  0 0 0 0 0 1 ;
      TR N 1  1 2  .03 .002   0 0 -7 ;  ENDWHILE ;
   ENDPROCEDURE ; RUN ; END ;
```

5.7 Introducing New Elements

When looking into the physics part of COSY INFINITY, it becomes apparent that all particle optical elements described above are nothing but procedures written in the COSY language. Due to the openness of the approach, users can construct their own particle optical elements.

Here we want to show how a user can define his own particle optical element and work with it. As a first example, we begin with a skew quadrupole that is rotated against the regular orientation by the angle ϕ. The action of such a quad can be obtained by first rotating the map by $-\phi$, then let the quad act, and finally rotate back. All these steps are performed on the DA variable containing the momentary value of the transfer map, which is the global COSY array MAP. For the conversion of degrees to radians, the global COSY variable DEGRAD is used. Note that many important global variables of COSY are described in section 5.8.

```
INCLUDE 'COSY' ;
```

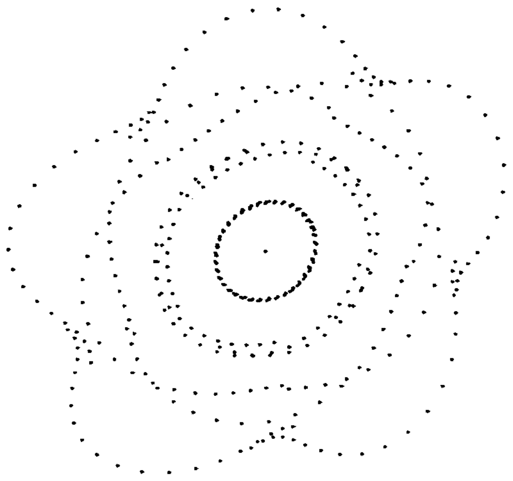


Figure 4: COSY LaTeX tracking picture

```
PROCEDURE RUN ;
   PROCEDURE SQ PHI L B D ;     {computes the action of a skew quad}
      PROCEDURE ROTATE PHI ;          {local procedure for rotation}
         VARIABLE M 1000 4 ; VARIABLE I 1 ;
         M(1) :=  COS(PHI*DEGRAD)*MAP(1) + SIN(PHI*DEGRAD)*MAP(3) ;
         M(3) := -SIN(PHI*DEGRAD)*MAP(1) + COS(PHI*DEGRAD)*MAP(3) ;
         M(2) :=  COS(PHI*DEGRAD)*MAP(2) + SIN(PHI*DEGRAD)*MAP(4) ;
         M(4) := -SIN(PHI*DEGRAD)*MAP(2) + COS(PHI*DEGRAD)*MAP(4) ;
         LOOP I 1 4 ; MAP(I) := M(I) ; ENDLOOP ; ENDPROCEDURE ;
      ROTATE -PHI ; MQ L B D ; ROTATE PHI ; ENDPROCEDURE ;
   OV 5 2 0 ; RP 1 1 1 ;
   UM ; DL .1 ; SQ -30 .2 .1 .1 ; DL .1 ; SQ 30 .2 .1 .1 ; PM 6 ;
   ENDPROCEDURE ; RUN ; END ;
```

It is clear that a similar technique can be used to study misaligned elements. In a similar way, it is easily possible to generate a "kick-environment" in COSY INFINITY, where every particle optical element is just represented by a kick in its center.

This technique is also useful in many other ways. For example, if a certain element is rather time consuming to compute, which can be the case with cylindrical lenses to high orders, one can write a procedure that computes the map of the element, including the dependence on some of its parameters, and saves the map somewhere. When called again with different values, the procedure decides if the values are close enough to the old ones to just utilize the previously computed map with the parameters plugged in, or if it is necessary to compute the element again. In case the parameters are varied only slightly, a very significant speed up can be achieved in this way, yet for the user the procedure looks like any other element.

5.8 Introducing New Features

The whole concept of COSY INFINITY is very open in that it easily allows extensions for specific tasks. The user is free to provide his own procedures for particle optical elements or for many other purposes. To interface with COSY INFINITY most efficiently, it is important to know the names of certain key global variables, functions and procedures. Furthermore it is important to know that all quantities in COSY INFINITY are in SI units, with the exception of voltages, which are in kV.

For some applications, it is helpful to access some of COSY INFINITY's global variables. Since the physics of the code is written in its own language, all these variables are directly visible to the user. The first set of relevant global variables are the natural constants describing the physics. These variables are set after the routine **RP** is called and can be utilized for calculations by the user. The data are taken from [29]. In order to match other codes, the variables can be changed by the user in COSY.FOX if necessary.

AMU	Atomic Mass Unit	$1.6605402 \cdot 10^{-27}$ kg
AMUMEV	Atomic Mass Unit in MeV	931.49432 MeV
EZERO	The charge unit	$1.60217733 \cdot 10^{-19}$ C
CLIGHT	The speed of light	$2.99792458 \cdot 10^{8}$ m/s
PI	the value of π	computed as 4 atan (1)

The second set of variables describes the reference particle. These variables are updated every time the procedure **RP** is called.

E0	Energy in MeV
M0	Mass in AMU
Z0	Charge in units
V0	Velocity
P0	Momentum
CHIM	Magnetic Rigidity
CHIE	Electric Rigidity
ETA	Kinetic Energy over mc^2

Finally, there are the variables that are updated by particle optical elements:

MAP	Array of 8 DA vectors containing Map
RAY	Array of 8 VE vectors containing Coordinates
SPOS	Momentary value of the independent variable

COSY INFINITY contains several procedures that are not used explicitly by the user but are used internally for certain operations. Firstly, there are the three DA functions

DER(<n>,<a>)

INTEG(<n>,<a>)

PB (<a>,<b>)

which compute the DA derivation with respect to variable n, the integral with respect to variable n, and the Poisson bracket between a and b. Another helpful function is

NMON (<NO>,<NV>)

which returns the maximum number of coefficients in a DA vector in NV variables to order NO. An important procedure is

POLVAL <L> <P> <NP> <A> <NA> <R> <NR> ;

which lets the polynomial described by the NP DA vectors stored in the array P act on the NA arguments A, and the result is stored in the NR Vectors R. The type of A is free, but all the array elements of A have to be the same type; it can be either DA or CD, in which case the procedure acts as a concatenator, it can be real, complex or intervals, in which case it acts like a polynomial evaluator, or it can be of vector type VE, in which case it acts as a very efficient vectorizing map evaluator and is used for repetitive tracking. If necessary, adding `0*A(1)` can make the type of the argument array element `A(I)` agreeing to that type of `A(1)`.

6 The COSY Language

6.1 General Aspects

In this section we will discuss the syntax of the COSY language. A brief summary of the commands can be found in section 6.6 on page 66. It will become apparent that the language has the flavor of PASCAL, which has a particularly simple syntax yet is relatively easy to analyze by a compiler and rather powerful.

The language of COSY differs from PASCAL in its object oriented features. New data types and operations on them can easily be implemented by putting them into a language description file described in the appendix. Furthermore, all type checking is done at run time, not at compile time. This has significant advantages for the practical use of DA and will be discussed below.

Throughout this section, curly brackets like "{" and "}" denote elements that can be repeated.

Most commands of the COSY language consist of a keyword, followed by expressions and names of variables, and terminated by a semicolon. The individual entries and the semicolon are separated by blanks. The exceptions are the assignment statement, which does not have a keyword but is identified by the assignment identifier :=, and the call to a procedure, in which case the procedure name is used instead of the keyword.

Line breaks are not significant; commands can extend over several lines, and several commands can be in one line. To facilitate readability of the code, it is possible to include comments. Everything contained within a pair of curly brackets "{" and "}" is ignored.

Each keyword and each name consist of up to 32 characters, of which the first has to be a letter and the subsequent ones can be letters, numbers or the underline sign "_". The case of the letters is not significant.

6.2 Program Segments and Structuring

The language consists of a tree-structured arrangement of nested program segments. There are three types of program segments. The first is the main program, of which there has to be exactly one and which has to begin at the top of the input file and ends at the end. It is denoted by the keywords

BEGIN ;

and

END ;

The other two types of program segments are procedures and functions. Their beginning and ending are denoted by the commands

PROCEDURE <name> { <name> } ;

and

ENDPROCEDURE ;

as well as

FUNCTION <name> { <name> } ;

ENDFUNCTION ;

The first name identifies the procedure and function for the purpose of calling it. The optional names define the local names of variables that are passed into the routine. Like in other languages, the name of the function can be used in arithmetic expressions, whereas the call to a procedure is a separate statement. Procedures and functions must contain at least one executable statement.

Inside each program segment, there are three sections. The first section contains the declaration of local variables, the second section contains the local procedures and functions, and the third section contains the executable code. A variable is declared with the following command:

VARIABLE <name> <expression> { <expression> } ;

Here the name denotes the identifier of the variable to be declared. As mentioned above, the types of variables are free at declaration time. The next expression contains the amount of memory that has to be allocated when the variable is used. The amount of memory has to be sufficient to hold the various types that the variable can assume. A real or double precision number number requires a length of 1, a complex double precision number a length of 2. A DA vector requires a length of at least the number of derivatives to be stored, and a CD vector requires twice that. The maximum number of derivatives through nth order in v variables can be obtained using the function **NMON**(n,v). Note that during allocation, the type and value of the variable is set to the real zero.

If the variable is to be used with indices as an array, the next expressions have to specify the different dimensions. Different elements of an array can have different types. This also allows to emulate most of the record concept found in PASCAL using arrays. As an example, the command

```
VARIABLE X 100 5 7 ;
```

declares X to be a two dimensional array with 5 respectively 7 entries, each of which has room for 100 memory locations.

Note that different from PASCAL practice, names of variables that are being passed into a function or procedure do not have to be declared.

All variables are visible inside the program segment in which they are declared as well as in all other program segments inside it. In case a variable has the same name as one that is visible from a higher level routine, its name and dimension override the name and properties of the higher level variable of the same name for the remainder of the procedure and all local procedures.

The next section of the program segment contains the declaration of local procedures and functions. Any such program segment is visible in the segment in which it was declared and in all program segments inside the segment in which it was declared, as long as the reference is physically located below the declaration of the local procedure. Recursive calls are permitted. Altogether, the local and global visibility of variables and procedures follows standard structured programming practice.

The third and final section of the program segment contains executable statements. Among the permissible executable statements is the assignment statement, which has the form

<variable or array element> := <expression> ;

The assignment statement does not require a keyword. It is characterized by the assignment identifier :=. The expression is a combination of variables and array elements visible in the routine, combined with operands and grouped by parentheses, following common practice. Note that due to the object oriented features, various operands can be loaded for various data types, and default hierarchies for the operands can be given. Parentheses are allowed to override default hierarchies. The indices of array elements can themselves be expressions.

Another executable statement is the call to a procedure. This statement does not require a keyword either. It has the form

<**procedure name**> { <expression> } ;

The name is the identifier of the procedure to be called which has to be visible at the current position. The rest are the arguments passed into the procedure. The number of arguments has to match the number of arguments in the declaration of the procedure.

6.3 Flow Control Statements

Besides the assignment statement and the procedure statement, there are statements that control the program flow. These statements consist of matching pairs denoting the beginning and ending of a control structure and sometimes of a third statement that can occur between such beginning and ending statements. Control statements can be nested as long as the beginning and ending of the lower level control structure is completely contained inside the same section of the higher level control structure.

The first such control structure begins with

IF <expression> ;

which later has to be matched by the command

ENDIF ;

If desired, there can be an arbitrary number of statements of the form

ELSEIF <expression> ;

between the matching **IF** and **ENDIF** statements.

If there is a structure involving **IF**, **ELSEIF** and **ENDIF**, the first expression in the **IF** or **ELSEIF** is evaluated. If it is not of logical type, an error message will be issued. If the value is true, execution will continue after the current line and until the next **ELSEIF**, at which point execution continues after the **ENDIF**.

If the value is false, the same procedure is followed with the logical expression in the next **ELSEIF**, until all of them have been reached, at which point execution continues after the **ENDIF**. So at most one of the sections of code separated by **IF** and the matching optional **ELSEIF** and the **ENDIF** statements is executed.

Note that there is nothing equivalent of a FORTRAN ELSE statement in the COSY language, but the same effect can be achieved with the statement ELSEIF 1=1 ;

The next such control structure consists of the pair

WHILE <expression> ;

and

ENDWHILE ;

If the expression is not of type logical, an error message will be issued. Otherwise, if it has the value true, execution is continued after the **WHILE** statement; otherwise, it is continued after the **ENDWHILE** statement. In the former case, execution continues until the **ENDWHILE** statement is reached. After this, it continues at the matching **WHILE**, where again the expression is checked. Thus, the block is run through over and over again as long as the expression has the proper value.

Another such control structure is the familiar loop, consisting of the pair

LOOP <name> <expression> <expression> {<expression>} ;

and

ENDLOOP ;

Here the first entry is the name of a visible variable which will act as the loop variable, the first and second expressions are the first and second bounds of the loop variable. If a third expression is present, this is the step size; otherwise, the step size is set to 1. Initially the loop variable is set to the first bound.

If the step size is positive or zero and the loop variable is not greater than the second bound, or the step size is negative and the loop variable is not smaller than the second bound, execution is continued at the next statement, otherwise after the matching **ENDLOOP** statement. When the matching **ENDLOOP** statement is reached after execution of the statements inside the loop, the step size is added to the loop variable. Then, the value of the loop variable is compared to the second bound in the same way as above, and execution is continued after the **LOOP** or the **ENDLOOP** statement, depending on the outcome of the comparison. Note that it is allowed to alter the value of the loop variable inside the loop, which allows a premature truncation of the loop.

The final control structure in the syntax of the COSY language allows nonlinear optimization as part of the syntax of the language. This is an unusual feature not found in other languages, and it could also be expressed in other ways using procedure calls. But the great importance of nonlinear optimization in applications of the language and the clarity in the code that can be achieved with it seemed to justify such a step. The structure consists of the pair

FIT <name> {<name>} ;

and

ENDFIT <expression> <expression> <expression> {<expression>} ;

Here the names denote the visible variables that are being adjusted. The first expression is the tolerance to which the minimum is requested. The second expression is the maximum number of evaluations of the objective function permitted. If this number is set to zero, no optimization is performed and the commands in the fit block are executed only once. The third expression gives the number of the optimizing algorithm that is being used. For the various optimizing algorithms, see section 7.1. The following expressions are of real or integer type and denote the objective quantities, the quantities that have to be minimized.

This structure is run through over and over again, where for each pass the optimization algorithm changes the values of the variables listed in the **FIT** statement and attempts to minimize the objective quantity. This continues until the algorithm does not succeed in decreasing the objective quantity anymore by more than the tolerance or the allowed number of iterations has been exhausted. After the optimization terminates, the variables contain the values corresponding to the lowest value of the objective quantity encountered by the algorithm.

Note that it is possible to terminate execution at any time by calling the intrinsic procedure **QUIT**. The procedure has one argument which determines if system information is provided. If this is not desired, the value 0 is used.

6.4 Input and Output

The COSY language has provisions for fully formatted or unformatted I/O. All input and output is performed using the two fundamental routines

READ <expression> <name> ;

and

WRITE <expression> {<expression>} ;

The first expression stands for a unit number, where using common notation, unit 5 denotes the keyboard and unit 6 denotes the screen. Unit numbers can be associated with particular file names by using the **OPENF** and **CLOSEF** procedures, which can be found in the index.

In the **READ** command, the name denotes the variable to be read. If the information that is read is a legal format free number, the variable will be of real type and contain the value of the number. In any other case, the variable will be of type string and contain the text just read.

For the case of formatted input, this string can be analyzed using the functions

SS (<string variable>,<I1>,<I2>)

which returns the substring from position I1 to position I2, as well as the function

R (<string variable>,<I1>,<I2>)

which converts the string representation of the real number contained in the substring from position I1 to I2 to the real number.

There are also dedicated read commands for other data types; for example, DA vectors can be read with the procedure DAREA (see index), and graphics meta files can be read with the procedure GRREA (see index).

In the **WRITE** command, the expressions following the unit are the output quantities. Each quantity will be printed in a separate line. As described a few lines below, by using the utilities to convert reals to strings **SF** and **S** and the concatenation of strings, full formatted output is also possible.

Depending on the momentary type of the expression, the form of the output will be as follows.

Strings are printed character by character, if necessary over several lines with 80 characters per line, followed by a line feed.

Real numbers will be printed in the FORTRAN format G20.12, followed by a line feed. Complex numbers will be printed in the Form (R,I), where R and I are the real and imaginary parts which are printed in the FORTRAN format G17.9; the number is followed by a line feed.

Differential Algebraic numbers will be output in several lines. Each line contains the expansion coefficient, the order, and the exponents of the independent variables that describe the term. Vanishing coefficients are not printed. Complex Differential Algebraic variables are printed in a similar way, except instead of one real coefficient, the real and imaginary parts of the complex coefficient is shown. We note that it is also possible to print several DA vectors simultaneously such that the coefficients of each vector correspond to one column. This can be achieved with the intrinsic procedure DAPRV (see index) and is used for example for the output of transfer maps in the procedure **PM** (see index).

Vectors are printed componentwise such that five components appear per line in the format G14.7. As discussed above, this can be used to output several reals in one line.

Logicals are output as TRUE or FALSE followed by a line feed. Interval numbers are output in the form [L,U], where L and U are the lower and upper bounds which are output as G17.9. Graphics objects are output in the way described in section 7.2.

As described above, each quantity in the **WRITE** command is output in a new line. To obtain formatted output, there are utilities to convert real numbers to strings, several of which can be concatenated into one string and hence output in one line. The concatenation is performed with the string operator "&" described in the appendix. The conversion of a real number to a string can be performed with the procedure RECST described in the appendix, as well as with the more convenient COSY function

SF (<real variable>,<format string>)

which returns the string representation of the real variable using the FORTRAN format specified in the format string. There is also a simplified version of this function

S (<real variable>)

which uses the FORTRAN format G20.12.

Besides the input and output of variables at execution, there are also commands that allow to save and include code in compiled form. This allows later inclusion in another program without recompiling, and thus achieves a similar function as linking The command

SAVE <name> ;

saves the compiled code in a file with the extension 'bin'; <name> is a string containing the name of root of the file, including paths and disks. The command

INCLUDE <name> ;

includes the previously compiled code. The name follows the same syntax as in the SAVE command.

Each code may contain only one INCLUDE statement, and it has to be located at the very top of the file. The SAVE and INCLUDE statements allow to break the code into a chain of easily manageable pieces and decrease compilation times considerably.

6.5 Error Messages

COSY distinguishes between five different kinds of error messages which have different meanings and require different actions to correct the underlying problem. The five types of error messages are identified by the symbols `###`, `$$$`, `!!!`, `@@@` and `***`. In addition, there are informational messages, denoted by `---`. The meaning of the error messages is as follows:

`###`: This error message denotes errors in the syntax of the user input. Usually a short message describing the problem is given, including the command in error. If this is not enough information to remedy the problem, the file <inputfile>.lis can be consulted. It contains an element-by-element listing of the user input, including the error messages at the appropriate positions.

`$$$`: This error message denotes runtime errors in a syntactically correct user input. Circumstances under which it is issued include array bound violations, type violations, missing initialization of variables, exhaustion of the memory of a variable, and illegal operations like division by zero.

!!!: This error message denotes exhaustion of certain internal arrays in the compiler. Since the basis of COSY is FORTRAN which is not recursive and requires a fixed memory allocation, all arrays used in the compiler have to be previously declared. This entails that in certain cases of big programs etc., the upper limits of the arrays can be reached. In such a case the user is told which parameter has to be increased. The problem can be remedied by replacing the value of the parameter by a larger value and re-compiling. Note that all occurrences of the parameter have to be changed; usually the parameters occur under the same name in many subroutines.

@@@: This message describes a catastrophic error, and should never occur with any kind of user input, erroneous or not. It means that COSY has found an internal error in its code by using certain self checks. In the rare case that such an error message is encountered, the user is kindly asked to contact us and submit the user program.

***: This error message denotes errors in the use of COSY INFINITY library procedures. It includes messages about improper sequences and improper values for parameters.

In case execution cannot be continued successfully, a system error exit is produced by deliberately attempting to compute the square root of -1.D0. Depending on the system COSY is run on, this will produce information about the status at the time of error. In order to be system independent, this is done by attempting to execute the computation of the root of a negative number.

6.6 List of Keywords

As a summary, below follows a complete list of keywords of the COSY language.

BEGIN		END
VARIABLE		
PROCEDURE		ENDPROCEDURE
FUNCTION		ENDFUNCTION
IF	ELSEIF	ENDIF
WHILE		ENDWHILE
LOOP		ENDLOOP
FIT		ENDFIT
WRITE	READ	
SAVE	INCLUDE	

7 Optimization and Graphics

7.1 Optimization

Many problems in the design of particle optical systems require the use of nonlinear optimization algorithms. COSY INFINITY supports the use of nonlinear optimizers at its language level using the commands **FIT** and **ENDFIT** (see section 6 beginning on page 58). The optimizers for this purpose are given as FORTRAN subroutines. For a list of momentarily available optimizers, see section 7.1.1. Because of a relatively simple interface, it is also possible to include new optimizers relatively easily. Details can be found in section 7.1.2.

Besides the FORTRAN algorithms for nonlinear optimization, the COSY language allows the user to design his own optimization strategies depending on the problem. Some thoughts about problem dependent optimizers can be found in section 7.1.3.

7.1.1 Optimizers

The **FIT** and **ENDFIT** commands of COSY allow the use of various different optimizers supplied in FORTRAN to minimize an objective function that depends on several variables. For details on the syntax of the commands, we refer to section 6 beginning on page 58. The choice of the proper objective function is up to the user, and it sometimes influences the success or failure of the optimization attempt.

While many problems are directly minimizations of a single intuitive quantity, others often require the simultaneous satisfaction of a set of conditions. In this later case, it is clearly possible to construct a total objective function that has a minimum if and only if the conditions are satisfied; this can for example be achieved by writing the conditions in the form $f_i(\vec{x}) = 0$ and then forming the objective function as a sum of absolute values or a sum of squares. By using factors in front of the summands, it is possible to give more or less emphasis on certain conditions.

At the present time, COSY supports three different optimizers with different features and strengths and weaknesses to minimize such objective functions. In the following we present a list of the various optimizers with a short description of their strengths and weaknesses. Each number is followed by the optimizer it identifies.

1. The Simplex Algorithm
   This optimizer is suitable for rather general objective functions that do not have to satisfy any smoothness criteria. It is quite rugged and finds local (and often global) minima in a rather large class of cases. In simple cases, it requires more execution time than algorithms for almost linear functions. Because of its generality it is often the algorithm of choice.

2. The Parabolic Minimizer
   This optimizer is particularly suitable for functions that are nearly quadratic in the variables. For suitable objective functions, it is rather fast and reliable. It often also works for other functions, but with a lower success rate than the Simplex Algorithm and a degradation of speed. Note that the user has significant freedom in choosing the objective function, and it is often possible to formulate the problem such that it can be represented by a nearly quadratic function.

3. The Simulated Annealing Algorithm
   This algorithm is often able to find the total minimum for very complicated objective functions, especially in cases where all other optimizers fail. This comes at the expense of a very high and often prohibitive number of function evaluations. Often this algorithm is also helpful for finding start values for the subsequent use of other algorithms.

4. The LMDIF optimizer
   This optimizer is a generalized least squares Newton method and very fast if it works, but not as robust as the simplex algorithm. For most cases, it should be the first optimizer to try.

7.1.2 Adding an Optimizer

COSY INFINITY has a relatively simple interface that allows the addition of other FORTRAN optimizers. All optimizers that can be used in COSY must use "reverse communication". This means that the optimizer does not control the program flow, but rather acts as an oracle which is called repeatedly. Each time it returns a point and requests that the objective function be evaluated at this new point, after which the optimizer is to be called again. This continues until the optimum is found, at which time a control variable is set to a certain value.

All optimizers are interfaced to COSY INFINITY via the routine FIT at the beginning of the file FOXFIT, which is the routine that is called from the code executer in FOXY. The arguments for the routine are as follows:

IFIT	$\rightarrow$	identification number of optimizer
XV	$\leftrightarrow$	current array of variables
NV	$\rightarrow$	number of variables
EPS	$\rightarrow$	desired accuracy of function value
ITER	$\rightarrow$	maximum allowed iteration number
IEND	$\leftarrow$	status identifier

The last argument, the status identifier, communicates the status of the optimization process to the executer of COSY. As long as it is nonzero, the optimizer requests evaluation of the objective function at the returned point X. If it is zero, the optimum has been found up to the abilities of the optimizer, and X contains the point where the minimum occurs.

The subroutine FIT branches to the various supported optimizers according to the value IFIT. It also supplies the various parameters required by the local optimizers. To include a new optimizer merely requires to put another statement label into the computed GOTO statement and to call the routine with the proper parameters.

We note that when writing an optimizer for reverse communication, it is very important to have the optimizer remember the variables describing the optimization status from one call to the next. This can be achieved using the FORTRAN statement SAVE. If the optimizer can return at several different positions, it is also important to retain the information from where the return occurred.

In case a user interfaces an optimizer of his own into COSY, we would appreciate receiving a copy of the amended file FOXFIT in order to be able to distribute the optimizer to other users as well.

7.1.3 Problem Dependent Optimization Strategies

In the case of very complicated optimization tasks, the use of any optimization algorithm alone is often too restrictive and inefficient. In practice it is more often than not necessary to combine certain "hand-tuning" tasks with the use of optimizers or to just check the terrain by evaluating the objective function on a coarse multidimensional grid.

We want to point out that because of its language structure, COSY INFINITY gives the demanding user very far reaching freedom to taylor his own optimization strategy. This can be achieved by properly nested structures involving loops, while blocks or if blocks. Manual tuning can be performed by reading certain variables from the screen in a while loop and then printing other quantities of interest or even the system graphics to the screen.

Besides being powerful, these strategies have also proved quite economical. In most cases it is possible to reduce the number of required runs considerably by choosing appropriate combinations of interactive tuning and hard wired as well as self made optimization strategies. With some advance thought this at least in principle makes it possible to design even rather complicated systems with only one COSY run.

7.2 Graphics

The object oriented language on which COSY INFINITY is based supports graphics via the graphics object. This is used for all the graphics generated by COSY and allows a rather elegant generation and manipulation of pictures.

The operand "&" allows the merging of graphics objects, and COSY INFINITY has functions that return individual moves and draws and various other elementary operations which can be glued together with "&". For details, we refer to the appendix beginning on page 78.

7.2.1 Simple Pictures

There are a few utilities that facilitate the interactive generation of pictures. The following command generates a frame, coordinate system, title, and axis marks:

FG <PIC> <XL> <XR> <YB> <YT> <DX> <DY> <TITLE> <I> ;

where PIC is a variable that has to be allocated by the user and that will contain the frame after the call. XL, XR, YB, YT are the x coordinates of the left and right corners and the y coordinates of the bottom and top corners. DX and DY are the distances between axis ticks in x and y directions. TITLE is a string containing the title or any other text that is to be displayed. I=0 produces a frame with aspect ratio 1.5 to 1 which fills the whole picture, whereas I=1 produces a square frame.

There is also a procedure that allows drawing simple curves:

CG <PIC> <X> <Y> <N> ;

where PIC is again the variable containing the picture, and X and Y are arrays with N coordinates describing the corner of the polygon. Note that it is necessary to produce a frame with **FG** before calling this routine.

7.2.2 Supported Graphics Drivers

COSY INFINITY allows to output graphics objects with a variety of drivers which are addressed by different unit numbers. A graphics object is output like any other variable in the COSY language using COSY's **WRITE** command. The different unit numbers correspond to the following drivers:

- positive: Low-Resolution ASCII output to respective unit; 6: screen.
- -1: GKS-based output to primary VMS/UIS workstation window
- -2: GKS-based Tektronix output
- -3: GKS-based X-Windows workstation output
- -4: GKS-based postscript output to PICTURE.PS
- -5: GKS-based HP 7475 plotter output to file HP7475.PLT

- -6: GKS-based HP Paintjet / DEC JL250 to file HPPAINT.PLT
- -7: Direct LaTeX picture mode output to files lpic001.tex, lpic002.tex, ...
- -8: VGA graphics output with Lahey F77 compiler for DOS/Windows
- -9: Direct Tektronix output without external graphics library
- -10: Direct PostScript output to files pic001.ps, pic002.ps, ... The header of each PostScript file contains the information about how to convert to an Encapsulated PostScript (EPS) file and how to include the picture in a LaTeX document.
- -11: Direct output to the low level graphics meta file METAGRAF.DAT
- -12: graPHIGS-based X-Windows workstation output
- -31 ... -41: VGA graphics output with Lahey F90 compiler for DOS/Windows
- -51 ... -60: GKS-based secondary VMS/UIS workstation windows output
- -61 ... -70: GKS-based secondary X-Windows workstation windows output
- -71 ... -80: GKS-based secondary Tektronix output
- -81 ... -90: graPHIGS-based secondary X-Windows workstation windows output
- -101 ... -110: PGPLOT X-Windows workstation or Windows PC windows output
- -111: PGPLOT output to PostScript files pgpic001.ps, pgpic002.ps, ...
- -112: PGPLOT output to LaTeX files pgpic001.tex, pgpic002.tex, ...

Positive unit numbers produce a low resolution 80 column by 24 lines ASCII output of the picture written to the respective unit, where unit 6 again corresponds to the screen.

Note that the following units require linking to the specific graphics packages.

- -1 ... -6, -51 ... -71: GKS package
  As reported by many users, unfortunately GKS is not as standardized as desirable, and often adaptations to the local implementation may be necessary.
  If linking to GKS package is not desired, the GKS driver routines in FOXGRAF.FOP should be removed and replaced by the provided dummy routines.
- -101 ... -112: PGPLOT package
  The PGPLOT Graphics Library is freely available from the PGPLOT web page, http://astro.caltech.edu/~tjp/pgplot/. Download and install the libraries according to the provided documentation on the target platform. Set the environment

variables accordingly. A sample makefile in page 8 shows how to link to the PGPLOT libraries.
If linking to PGPLOT package is not desired, the GKS driver routines in FOXGRAF.FOP should be removed and replaced by the provided dummy routines.

- -8: VGA graphics package with Lahey F77 compiler for DOS/Windows
  The lines containing the graphics package specific description in FOXGRAF.FOP are commented out. Uncomment all the lines containing the string *L77 in columns 73 to 80 in FOXGRAF.FOP.

- -31 ... -41: VGA graphics package with Lahey F90 compiler for DOS/Windows
  The lines containing the graphics package specific description in FOXGRAF.FOP are commented out. Uncomment all the lines containing the string *L90 in columns 73 to 80 in FOXGRAF.FOP.

- -12, -81 ... -91: graPHIGS package
  The lines containing the graphics package specific description in FOXGRAF.FOP are commented out. Uncomment all the lines containing the string *PHG in columns 73 to 80 in FOXGRAF.FOP.

The other graphics drivers are self-contained within COSY.

Graphics written to a meta file can be read from a unit to a variable PIC with the command

GRREAD <unit> <PIC> ;

7.2.3 Adding Graphics Drivers

To facilitate the adaptation to new graphics packages, COSY INFINITY has a very simple standardized graphics interface in the file FOXGRAF.FOP. In order to write drivers for a new graphics package, the user has to supply a set of seven routines interfacing to the graphics package. For ease of identification and uniformity, the names of the routines should begin with a three letter identifier for the graphics system, and should end with three letters identifying their task. The required routines are

1. ...BEG : Begins the picture. Allows calling all routines necessary to initiate a picture.

2. ...MOV(X,Y) : Performs a move of the pen to coordinates X,Y. Coordinates range from 0 to 1.

3. ...DRA(X,Y) : Performs a draw from the current position to coordinates X,Y. Coordinates range from 0 to 1.

4. ...DOT(X,Y) : Performs a move of the pen to coordinates X,Y, then prints a dot at the position. Coordinates range from 0 to 1.
5. ...CHA(I) : Prints ASCII character I to momentary position. Size of the character has to be 1/80 of the total picture size. After the character, the position is advanced in X direction by 1/80.
6. ...COL(I) : Sets a color. If supported by the system, the colors are referred to by the following integers: 1: black, 2: blue, 3: red, 4: yellow, 5: green.
7. ...WID(I) : Sets the width of the pen. I ranges from 1 to 10, 1 denoting the finest and 10 the thickest line.
8. ...END : Concludes the picture. Allows calling all routines necessary to close the picture and print it.

The arguments X and Y are DOUBLE PRECISION, and I is INTEGER. After these routines have been created, the routine GRPRI in FOXGRAF.FOP has to be modified to include calls to the above routines at positions where the other corresponding routines are called for other graphics standards.

We appreciate receiving drivers for other graphics systems written by users to include them into the master version of the code.

7.2.4 The COSY Graphics Meta File

In case it is not desired to write driver routines at the FORTRAN level, it is possible to utilize the COSY graphics meta file, which is written in ASCII to the file METAGRAF.DAT via unit -11. This meta file can be easily read by standard Graphics programs such as TOPDRAWER, DISPLA, or GNUPLOT, or by programs written by users.

The meta file consists of a list of elementary operations discussed in the last subsection. Each of these seven elementary operations is output in a separate line, where the first three characters identify the command, then follows a blank, and then the parameters. The positions X and Y are output as 2E24.16, the character A as A1, and the numbers I as I1.

```
BEG
MOV X Y
DRA X Y
CHA A
COL I
WID I
END
```

8 Acknowledgements

For very valuable help with an increasing number of parts of the program, I would like to thank Meng Zhao, Weishi Wan, Georg Hoffstätter, Ralf Degenhardt, Khodr Shamseddine, Kyoko Makino, Nina Golubeva, Vladimir Balandin, Jens Hoefkens, Béla Erdélyi, Lars Diening, Michael Lindemann, and Jens von Bergmann, who all at various times were at Michigan State University. I would also like to thank numerous COSY users for providing valuable feedback, many good suggestions, and streamlining the implementation on various machines.

I would like to thank Kurt Overley and Tom Mottershead for their optimizer QOPT, Jay Dittmann for help with the simplex optimizer, and Jorge More for providing the public domain optimizer LMDIF. I would like to thank Roger Servranckx for providing the MAD to COSY converter. I would like to thank Felix Marti for writing the GKS graphics drivers.

Financial support was appreciated from the Deutsche Forschungsgemeinschaft, the University of Gießen, the SSC Central Design Group, Lawrence Berkeley Laboratory, Michigan State University, the National Superconducting Cyclotron Laboratory, the Alfred P. Sloan Foundation, and the U.S. Department of Energy.

References

[1] M. Berz. Computational aspects of design and simulation: COSY INFINITY. *Nuclear Instruments and Methods*, A298:473, 1990.

[2] M. Berz. COSY INFINITY, an arbitrary order general purpose optics code. *Computer Codes and the Linear Accelerator Community*, Los Alamos LA-11857-C:137, 1990.

[3] K. L. Brown. The ion optical program TRANSPORT. Technical Report 91, SLAC, 1979.

[4] T. Matsuo and H. Matsuda. Computer program TRIO for third order calculations of ion trajectories. *Mass Spectrometry*, 24, 1976.

[5] H. Wollnik, J. Brezina, and M. Berz. GIOS-BEAMTRACE, a computer code for the design of ion optical systems including linear or nonlinear space charge. *Nuclear Instruments and Methods*, A258:408, 1987.

[6] M. Berz, H. C. Hofmann, and H. Wollnik. COSY 5.0, the fifth order code for corpuscular optical systems. *Nuclear Instruments and Methods*, A258:402, 1987.

[7] M. Berz and H. Wollnik. The program HAMILTON for the analytic solution of the equations of motion in particle optical systems through fifth order. *Nuclear Instruments and Methods*, A258:364, 1987.

[8] M. Berz. Arbitrary order description of arbitrary particle optical systems. *Nuclear Instruments and Methods*, A298:426, 1990.

[9] M. Berz. Differential algebraic description of beam dynamics to very high orders. *Particle Accelerators*, 24:109, 1989.

[10] M. Berz. Differential algebraic treatment of beam dynamics to very high orders including applications to spacecharge. *AIP Conference Proceedings*, 177:275, 1988.

[11] M. Berz. *The Description of Particle Accelerators using High Order Perturbation Theory on Maps, in: M. Month (Ed), Physics of Particle Accelerators*, volume 1, page 961. American Institute of Physics, 1989.

[12] M. Berz. High-order description of accelerators using differential algebra and first applications to the SSC. In *Proceedings, Snowmass Summer Meeting*, Snowmass, Co, 1988.

[13] E. Forest and M. Berz. *Canonical Integration and Analysis of Periodic Maps using Non-Standard Analysis and Lie Methods, in: Lie Methods in Optics II*, pages 47–66. Springer, Berlin, 1989.

[14] M. Berz. Differential algebra precompiler version 3 reference manual. Technical Report MSUCL-755, Michigan State University, East Lansing, MI 48824, 1990.

[15] E. Forest, M. Berz, and J. Irwin. Normal form methods for complicated periodic systems: A complete solution using Differential algebra and Lie operators. *Particle Accelerators*, 24:91, 1989.

[16] H. Wollnik, B. Hartmann, and M. Berz. Principles behind GIOS and COSY. *AIP Conference Proceedings*, 177:74, 1988.

[17] B. Hartmann, M. Berz, and H. Wollnik. The computation of fringing fields using Differential Algebra. *Nuclear Instruments and Methods*, A297:343, 1990.

[18] B. Hartmann, H. Wollnik, and M. Berz. TRIBO, a program to determine high-order properties of intense ion beams. *Computer Codes and the Linear Accelerator Community*, Los Alamos LA-11857-C:431, 1990.

[19] H. Wollnik, J. Brezina, and M. Berz. GIOS-BEAMTRACE, a program for the design of high resolution mass spectrometers. In *Proceedings AMCO-7*, page 679, Darmstadt, 1984.

[20] A. J. Dragt, L. M. Healy, F. Neri, and R. Ryne. MARYLIE 3.0 - a program for nonlinear analysis of accelerators and beamlines. *IEEE Transactions on Nuclear Science*, NS-3,5:2311, 1985.

[21] C. Iselin. MAD - a reference manual. Technical Report LEP-TH/85-15, CERN, 1985.

[22] C. Iselin and J. Niederer. The MAD program, version 7.2, user's reference manual. Technical Report CERN/LEP-TH/88-38, CERN, 1988.

[23] H. Wollnik. *Optics of Charged Particles.* Academic Press, Orlando, Florida, 1987.

[24] P. W. Hawkes and E. Kasper. *Principles of Electron Optics*, volume 1-2. Academic Press, London, 1989.

[25] D. C. Carey. *The Optics of Charged Particle Beams.* Harwood Academic, New York, 1987, 1992.

[26] X. Jiye. *Aberration Theory in Electron and Ion Optics.* Advances in Electronics and Electron Physics, Supplement 17. Academic Press, Orlando, Florida, 1986.

[27] K. G. Steffen. *High Energy Beam Optics.* Wiley-Interscience, New York, 1965.

[28] W. Glaser. *Grundlagen der Elektronenoptik.* Springer, Wien, 1952.

[29] E. R. Cohen and B. N. Taylor. 1986 adjustment to the fundamental physics constants. *Review Modern Physics*, 59:1121, 1987, revised 1989.

[30] S. Kowalski and H. Enge. RAYTRACE. Technical report, MIT, Cambridge, Massachussetts, 1985.

[31] K. L. Brown and J. E. Spencer. Non-linear optics for the final focus of the single-pass-collider. *IEEE Transactions on Nuclear Science*, NS-28,3:2568, 1981.

[32] G. Hoffstätter and M. Berz. Efficient computation of fringe fields using symplectic scaling. In *Third Computational Accelerator Physics Conference*, page 467. AIP Conference Proceedings 297, 1993.

[33] G. Hoffstätter and M. Berz. Symplectic scaling of transfer maps including fringe fields. *Physical Review E*, 54,4, 1996.

[34] K. Makino and M. Berz. Arbitrary order aberrations for elements characterized by measured fields. In *Proc. SPIE*, volume 3155, pages 221–227. SPIE, 1997.

[35] K. Makino. *Rigorous Analysis of Nonlinear Motion in Particle Accelerators.* PhD thesis, Michigan State University, East Lansing, Michigan, USA, 1998. also MSUCL-1093.

[36] M. Berz, K. Joh, J. A. Nolen, B. M. Sherrill, and A. F. Zeller. Reconstructive correction of aberrations in nuclear particle spectrographs. *Physical Review C*, 47,2:537, 1993.

[37] M. Berz, K. Joh, J. A. Nolen, B. M. Sherrill, and A. F. Zeller. Reconstructive correction of aberrations in particle spectrographs. Technical Report MSUCL-812, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI 48824, 1991.

[38] M. Berz. Differential algebraic formulation of normal form theory. In *M. Berz, S. Martin and K. Ziegler (Eds.), Proc. Nonlinear Effects in Accelerators*, page 77. IOP Publishing, 1992.

[39] M. Berz. Direct computation and correction of chromaticities and parameter tune shifts in circular accelerators. In *Proceedings XIII International Particle Accelerator Conference, JINR D9-92-455*, pages 34–47(Vol.2), Dubna, 1992.

[40] G. H. Hoffstätter. *Rigorous bounds on survival times in circular accelerators and efficient computation of fringe-field transfer maps.* PhD thesis, Michigan State University, East Lansing, Michigan, USA, 1994. also DESY 94-242.

[41] M. Berz and G. Hoffstätter. Exact bounds of the long term stability of weakly nonlinear systems applied to the design of large storage rings. *Interval Computations*, 2:68–89, 1994.

[42] A. J. Dragt and J. M. Finn. Lie series and invariant functions for analytic symplectic maps. *Journal of Mathematical Physics*, 17:2215, 1976.

[43] A. J. Dragt. Lectures on nonlinear orbit dynamics. In *1981 Fermilab Summer School.* AIP Conference Proceedings Vol. 87, 1982.

[44] M. Berz. Isochronous beamlines for free electron lasers. *Nuclear Instruments and Methods*, A298:106, 1990.

[45] M. Berz. Differential algebraic description and analysis of spin dynamics. In *Proceedings, SPIN94*, 1995.

[46] V. Balandin, M. Berz, and N. Golubeva. Computation and analysis of spin dynamics. In *Fourth Computational Accelerator Physics Conference*, volume 391, page 276. AIP Conference Proceedings, 1996.

[47] W. Wan. *Theory and Applications of Arbitrary-Order Achromats.* PhD thesis, Michigan State University, East Lansing, Michigan, USA, 1995. also MSUCL-976.

[48] W. Wan and M. Berz. An analytical theory of arbitrary order achromats. *Physical Review E*, 54,3:2870, 1996.

9 Appendix: The Supported Types and Operations

The language of COSY INFINITY is object oriented, and it is very simple to create new data types and define operations on them. Details about the types and operations are described in the language description data file GENFOX.DAT which is read by the program GENFOX, which then updates the source code of the compiler.

The first entry in GENFOX.DAT is a list of the names of all data types. The second entry is a list containing the elementary operations, information for which combinations of data types are allowed, and the names of individual FORTRAN routines to perform the specific operations.

The third entry contains all the intrinsic functions and the types of their results. The fourth entry finally contains a list of FORTRAN procedures that can be called from the environment.

All these data are read from a program that updates the compiler; in particular, it includes all the intrinsic operations, functions and procedures into the routine that interprets the intermediate code.

Below follows a list of all object types as well as a list of all the operands available for various combinations of objects, the available intrinsic functions, and the available intrinsic procedures. These lists are automatically produced in LaTeX format by the program GENFOX with each compiler update.

This information is current as of 28-Jan-00.

In this version, the following types are supported:

RE	8 Byte Real Number
ST	String
LO	Logical
DA	Differential Algebra Vector
VE	Real Number Vector
CM	8 Byte Complex Number
IN	8 Byte Interval Number
IV	Interval Vector
OI	Ordered Interval Number
OV	Ordered Interval Vectors
GR	Graphics
CD	Complex Differential Algebra Vector
RD	Remainder-enhanced Differential Algebra Object

Now follows a list of all operations available for various combinations of types. For each operation, a relative priority (Pr.) is given which determines the hierarchy of the operations in expressions if there are no parentheses.

Operation	Pr.	Type			Comment
		Left	Right	Result	
+	3	RE	RE	RE	
+	3	RE	DA	DA	
+	3	RE	VE	VE	
+	3	RE	CM	CM	
+	3	RE	IN	IN	
+	3	RE	IV	IV	
+	3	RE	OI	OI	
+	3	RE	OV	OV	
+	3	RE	CD	CD	
+	3	RE	RD	RD	
+	3	LO	LO	LO	logical OR
+	3	DA	RE	DA	
+	3	DA	DA	DA	
+	3	DA	CM	CD	
+	3	DA	CD	CD	
+	3	VE	RE	VE	
+	3	VE	VE	VE	
+	3	CM	RE	CM	
+	3	CM	DA	CD	
+	3	CM	CM	CM	
+	3	CM	CD	CD	
+	3	IN	RE	IN	
+	3	IN	IN	IN	
+	3	IV	RE	IV	
+	3	IV	IV	IV	
+	3	OI	RE	OI	
+	3	OI	OI	OI	
+	3	OV	RE	OV	
+	3	OV	OV	OV	
+	3	CD	RE	CD	
+	3	CD	DA	CD	
+	3	CD	CM	CD	
+	3	CD	CD	CD	
+	3	RD	RE	RD	
+	3	RD	RD	RD	

Operation	Pr.	Type			Comment
		Left	Right	Result	
-	3	RE	RE	RE	
-	3	RE	DA	DA	
-	3	RE	VE	VE	
-	3	RE	CM	CM	
-	3	RE	IN	IN	
-	3	RE	IV	IV	
-	3	RE	OI	OI	
-	3	RE	OV	OV	
-	3	RE	CD	CD	
-	3	RE	RD	RD	
-	3	DA	RE	DA	
-	3	DA	DA	DA	
-	3	DA	CM	CD	
-	3	DA	CD	CD	
-	3	VE	RE	VE	
-	3	VE	VE	VE	
-	3	CM	RE	CM	
-	3	CM	DA	CD	
-	3	CM	CM	CM	
-	3	CM	CD	CD	
-	3	IN	RE	IN	
-	3	IN	IN	IN	
-	3	IV	RE	IV	
-	3	IV	IV	IV	
-	3	OI	RE	OI	
-	3	OI	OI	OI	
-	3	OV	RE	OV	
-	3	OV	OV	OV	
-	3	CD	RE	CD	
-	3	CD	DA	CD	
-	3	CD	CM	CD	
-	3	CD	CD	CD	
-	3	RD	RE	RD	
-	3	RD	RD	RD	

Operation	Pr.	Type			Comment
		Left	Right	Result	
*	4	RE	RE	RE	
*	4	RE	DA	DA	
*	4	RE	VE	VE	
*	4	RE	CM	CM	
*	4	RE	IN	IN	
*	4	RE	IV	IV	
*	4	RE	OI	OI	
*	4	RE	OV	OV	
*	4	RE	CD	CD	
*	4	RE	RD	RD	
*	4	LO	LO	LO	logical AND
*	4	DA	RE	DA	
*	4	DA	DA	DA	
*	4	DA	CM	CD	
*	4	DA	CD	CD	
*	4	VE	RE	VE	
*	4	VE	VE	VE	
*	4	CM	RE	CM	
*	4	CM	DA	CD	
*	4	CM	CM	CM	
*	4	CM	CD	CD	
*	4	IN	RE	IN	
*	4	IN	IN	IN	
*	4	IV	RE	IV	
*	4	IV	IV	IV	
*	4	OI	RE	OI	
*	4	OI	OI	OI	
*	4	OV	RE	OV	
*	4	OV	OV	OV	
*	4	CD	RE	CD	
*	4	CD	DA	CD	
*	4	CD	CM	CD	
*	4	CD	CD	CD	
*	4	RD	RE	RD	
*	4	RD	RD	RD	

Operation	Pr.	Type			Comment
		Left	Right	Result	
/	4	RE	RE	RE	
/	4	RE	DA	DA	
/	4	RE	VE	VE	
/	4	RE	CM	CM	
/	4	RE	IN	IN	
/	4	RE	IV	IV	
/	4	RE	CD	CD	
/	4	RE	RD	RD	
/	4	DA	RE	DA	
/	4	DA	DA	DA	
/	4	DA	CM	CD	
/	4	DA	CD	CD	
/	4	VE	RE	VE	
/	4	VE	VE	VE	
/	4	CM	RE	CM	
/	4	CM	DA	CD	
/	4	CM	CM	CM	
/	4	CM	CD	CD	
/	4	IN	RE	IN	
/	4	IN	IN	IN	
/	4	IV	RE	IV	
/	4	IV	IV	IV	
/	4	OI	RE	OI	
/	4	OV	RE	OV	
/	4	CD	RE	CD	
/	4	CD	DA	CD	
/	4	CD	CM	CD	
/	4	CD	CD	CD	
/	4	RD	RE	RD	
/	4	RD	RD	RD	

Operation	Pr.	Type			Comment
		Left	Right	Result	
^	5	RE	RE	RE	
^	5	VE	RE	VE	
^	5	IN	RE	IN	
^	5	IV	RE	IV	

Operation	Pr.	Type			Comment
		Left	Right	Result	
<	2	RE	RE	LO	
<	2	ST	ST	LO	

Operation	Pr.	Type			Comment
		Left	Right	Result	
>	2	RE	RE	LO	
>	2	ST	ST	LO	

Operation	Pr.	Type			Comment
		Left	Right	Result	
=	2	RE	RE	LO	
=	2	ST	ST	LO	

Operation	Pr.	Type			Comment
		Left	Right	Result	
#	2	RE	RE	LO	
#	2	ST	ST	LO	

Operation	Pr.	Type			Comment
		Left	Right	Result	
&	2	RE	RE	VE	
&	2	RE	VE	VE	
&	2	ST	ST	ST	
&	2	VE	RE	VE	
&	2	VE	VE	VE	
&	2	IN	IN	IV	
&	2	IN	IV	IV	
&	2	IV	IN	IV	
&	2	IV	IV	IV	
&	2	GR	GR	GR	

Now follows a list of all intrinsic functions available in the momentary version with a short description and the allowed types.

- TYPE returns the type of an object as a number

Function	Argument Type	Type of Result
TYPE	RE	RE
TYPE	ST	RE
TYPE	LO	RE
TYPE	DA	RE
TYPE	VE	RE
TYPE	CM	RE
TYPE	IN	RE
TYPE	IV	RE
TYPE	OI	RE
TYPE	OV	RE
TYPE	GR	RE
TYPE	CD	RE
TYPE	RD	RE

- LENGTH returns the momentary length of a variable in 8 byte blocks

Function	Argument Type	Type of Result
LENGTH	RE	RE
LENGTH	ST	RE
LENGTH	LO	RE
LENGTH	DA	RE
LENGTH	VE	RE
LENGTH	CM	RE
LENGTH	IN	RE
LENGTH	IV	RE
LENGTH	OI	RE
LENGTH	OV	RE
LENGTH	GR	RE
LENGTH	CD	RE
LENGTH	RD	RE

- VARMEM returns the current memory address of an object

Function	Argument Type	Type of Result
VARMEM	RE	RE
VARMEM	ST	RE
VARMEM	LO	RE
VARMEM	DA	RE
VARMEM	VE	RE
VARMEM	CM	RE
VARMEM	IN	RE
VARMEM	IV	RE
VARMEM	OI	RE
VARMEM	OV	RE
VARMEM	GR	RE
VARMEM	CD	RE
VARMEM	RD	RE

- VARPOI returns the current pointer address of an object

Function	Argument Type	Type of Result
VARPOI	RE	RE
VARPOI	ST	RE
VARPOI	LO	RE
VARPOI	DA	RE
VARPOI	VE	RE
VARPOI	CM	RE
VARPOI	IN	RE
VARPOI	IV	RE
VARPOI	OI	RE
VARPOI	OV	RE
VARPOI	GR	RE
VARPOI	CD	RE
VARPOI	RD	RE

- NOT returns the negation of a logical

Function	Argument Type	Type of Result
NOT	LO	LO

- EXP computes the exponential function

Function	Argument Type	Type of Result
EXP	RE	RE
EXP	DA	DA
EXP	VE	VE
EXP	CM	CM
EXP	IN	IN
EXP	IV	IV
EXP	RD	RD

- LOG computes the natural logarithm

Function	Argument Type	Type of Result
LOG	RE	RE
LOG	DA	DA
LOG	VE	VE
LOG	CM	CM
LOG	IN	IN
LOG	IV	IV
LOG	RD	RD

- SIN computes the sine

Function	Argument Type	Type of Result
SIN	RE	RE
SIN	DA	DA
SIN	VE	VE
SIN	CM	CM
SIN	IN	IN
SIN	IV	IV
SIN	RD	RD

- COS computes the cosine

Function	Argument Type	Type of Result
COS	RE	RE
COS	DA	DA
COS	VE	VE
COS	CM	CM
COS	IN	IN
COS	IV	IV
COS	RD	RD

- TAN computes the tangent

Function	Argument Type	Type of Result
TAN	RE	RE
TAN	DA	DA
TAN	VE	VE
TAN	IN	IN
TAN	IV	IV
TAN	RD	RD

- ASIN computes the arc sine

Function	Argument Type	Type of Result
ASIN	RE	RE
ASIN	DA	DA
ASIN	VE	VE
ASIN	IN	IN
ASIN	IV	IV
ASIN	RD	RD

- ACOS computes the arc cosine

Function	Argument Type	Type of Result
ACOS	RE	RE
ACOS	DA	DA
ACOS	VE	VE
ACOS	IN	IN
ACOS	IV	IV
ACOS	RD	RD

- ATAN computes the arc tangent

Function	Argument Type	Type of Result
ATAN	RE	RE
ATAN	DA	DA
ATAN	VE	VE
ATAN	IN	IN
ATAN	IV	IV
ATAN	RD	RD

- SINH computes the hyperbolic sine

Function	Argument Type	Type of Result
SINH	RE	RE
SINH	DA	DA
SINH	VE	VE
SINH	CM	CM
SINH	IN	IN
SINH	IV	IV
SINH	RD	RD

- COSH computes the hyperbolic cosine

Function	Argument Type	Type of Result
COSH	RE	RE
COSH	DA	DA
COSH	VE	VE
COSH	CM	CM
COSH	IN	IN
COSH	IV	IV
COSH	RD	RD

- TANH computes the hyperbolic tangent

Function	Argument Type	Type of Result
TANH	RE	RE
TANH	DA	DA
TANH	VE	VE
TANH	IN	IN
TANH	IV	IV
TANH	RD	RD

- SQRT computes the square root

Function	Argument Type	Type of Result
SQRT	RE	RE
SQRT	DA	DA
SQRT	VE	VE
SQRT	CM	CM
SQRT	IN	IN
SQRT	IV	IV
SQRT	RD	RD

- ISRT computes the reciprocal of square root

Function	Argument Type	Type of Result
ISRT	RE	RE
ISRT	DA	DA
ISRT	VE	VE
ISRT	IN	IN
ISRT	IV	IV
ISRT	RD	RD

- SQR computes the square

Function	Argument Type	Type of Result
SQR	RE	RE
SQR	DA	DA
SQR	VE	VE
SQR	CM	CM
SQR	IN	IN
SQR	IV	IV
SQR	OI	OI
SQR	OV	OV
SQR	CD	CD
SQR	RD	RD

- ERF computes the real error function erf

Function	Argument Type	Type of Result
ERF	RE	RE
ERF	DA	DA

- WERF computes the complex error function w

Function	Argument Type	Type of Result
WERF	CM	CM
WERF	CD	CD

- ABS computes the absolute value

Function	Argument Type	Type of Result
ABS	RE	RE
ABS	DA	RE
ABS	VE	VE
ABS	CM	RE
ABS	CD	RE

- NORM computes the norm of a vector

Function	Argument Type	Type of Result
NORM	DA	RE
NORM	VE	VE
NORM	CD	RE

- CONS determines the constant part

Function	Argument Type	Type of Result
CONS	RE	RE
CONS	DA	RE
CONS	VE	RE
CONS	CM	CM
CONS	IN	RE
CONS	IV	VE
CONS	CD	CM
CONS	RD	RE

- RE determines the real number part

Function	Argument Type	Type of Result
RE	RE	RE
RE	DA	RE
RE	RD	RE

- IN computes the interval bound

Function	Argument Type	Type of Result
IN	RE	IN
IN	VE	IN
IN	IN	IN
IN	RD	IN

- WIDTH computes the width of an interval bound

Function	Argument Type	Type of Result
WIDTH	IN	RE
WIDTH	RD	RE

- REAL determines the real part

Function	Argument Type	Type of Result
REAL	RE	RE
REAL	DA	DA
REAL	CM	RE
REAL	CD	DA
REAL	RD	RD

- IMAG determines the imaginary part

Function	Argument Type	Type of Result
IMAG	RE	RE
IMAG	DA	DA
IMAG	CM	RE
IMAG	CD	DA
IMAG	RD	RD

- INT determines the integer part

Function	Argument Type	Type of Result
INT	RE	RE

- NINT determines the nearest integer

Function	Argument Type	Type of Result
NINT	RE	RE

- DA returns the i th elementary DA vector

Function	Argument Type	Type of Result
DA	RE	DA

- CMPLX converts to complex

Function	Argument Type	Type of Result
CMPLX	RE	CM
CMPLX	DA	CD
CMPLX	CM	CM
CMPLX	CD	CD

- CONJ conjugates a complex number

Function	Argument Type	Type of Result
CONJ	RE	RE
CONJ	DA	DA
CONJ	CM	CM
CONJ	CD	CD
CONJ	RD	RD

In addition to the just listed operators and intrinsic functions, the following intrinsic procedures are available:

- Procedure MEMALL (1 argument) returns the amount of memory allocated at this time.
- Procedure MEMFRE (1 argument) returns the amount of memory still available at this time.
- Procedure OPENF (3 arguments) opens a file. Parameters are unit number, filename (string), and status (string, same as in FORTRAN open).
- Procedure CLOSEF (1 argument) closes a file. Parameter is unit number.
- Procedure REWF (1 argument) rewinds a file. Parameter is unit number.
- Procedure BACKF (1 argument) backspaces a file. Parameter is unit number.
- Procedure CPUSEC (1 argument) returns the elapsed CPU time in seconds since last midnight on VMS; the elapsed system time on PC; and the elapsed CPU time in the process on UNIX.
- Procedure QUIT (1 argument) terminates execution; Argument = 1 triggers traceback.
- Procedure SCRLEN (1 argument) sets the amount of space scratch variables are allocated with.
- Procedure DAINI (4 arguments) initializes the order and number of variables of DA. Arguments are order, number of variables, output unit number, number of monomials (return).
- Procedure DANOT (1 argument) sets momentary truncation order for DA.
- Procedure DAEPS (1 argument) sets zero tolerance for components of DA vectors.
- Procedure DAPEW (4 arguments) prints the part of DA vector that has a certain order in a specified variable. Arguments are unit, DA vector, column number, and order.
- Procedure DAREA (3 arguments) reads a DA vector. Arguments are the unit number, the variable name and the number of independent variables.
- Procedure DAPRV (5 arguments) prints an array of DA vectors. Arguments are the array, the number of components, maximum and current main variable number, and the unit number.
- Procedure DAREV (5 arguments) reads an array of DA vectors. Arguments are the array, the number of components, maximum and current main variable number, and the unit number.

- Procedure DAFLO (4 arguments) computes the flow of x' = f(x) for time step 1. Arguments: the initial condition, array of right hand sides, result, and dimension of f.
- Procedure CDFLO (4 arguments) same as DAFLO but with complex arguments.
- Procedure RERAN (1 argument) returns a random number between -1 and 1.
- Procedure DARAN (2 arguments) fills DA vector with random entries. Arguments are DA vector and fill factor.
- Procedure DADIU (3 arguments) performs a division by a unit vector if possible. Arguments are the number of the unit vector and the two DA vectors.
- Procedure DADER (3 arguments) performs the derivation operation on DA vector. Arguments are the number with respect to which to differentiate and the two DA vectors.
- Procedure DAINT (3 arguments) performs an integration of a DA vector. Arguments are the number with respect to which to integrate and the two DA vectors.
- Procedure DAPLU (4 arguments) replaces power of independent variable I by constant C. Arguments are the DA or CD vector, I, C, and the resulting DA or CD vector.
- Procedure DAPEE (3 arguments) returns a component of a DA or CD vector. Arguments are the DA or CD vector, the id for the coefficient in TRANSPORT notation, and the returning real or complex number.
- Procedure DAPEA (4 arguments) performs same with DAPEE, except the id is specified by an array with each element denoting the exponent. The third argument is the size of the array.
- Procedure DAPEP (4 arguments) returns a parameter dependent component of a DA or CD vector. Arguments are the DA or CD vector, the coefficient id, number of variables, and the resulting DA or CD vector.
- Procedure DANOW (3 arguments) computes the order weighted norm of the DA vector in the first argument. Second argument: weight, third argument: result.
- Procedure CDF2 (5 arguments) Lets exp(: f_2 :)) act on first argument in Floquet variables. Other Arguments: 3 tunes (2π), result.
- Procedure CDNF (8 arguments) Lets $1/(1 - \exp(: f_2 :))$ act on first argument in Floquet variables. Other Arguments: 3 tunes (2π), array of resonances with dimensions, result.

- Procedure CDNFDA (7 arguments) Lets $C_j^{\pm}$ act on the first argument. Other Arguments: moduli, arguments, coordinate number, total number, epsilon, and result.
- Procedure CDNFDS (7 arguments) Lets $S_j^{\pm}$ act on the first argument. Other Arguments: moduli, arguments, spin argument, total number, epsilon, and result.

- Procedure MTREE (7 arguments) computes the tree representation of a DA array. Arguments: DA array, elements, coefficient array, 2 steering arrays, elements, length of tree.
- Procedure LINV (5 arguments) inverts a quadratic matrix. Arguments are the matrix, the inverse, the number of actual entries, the allocation dimension, and an error flag.
- Procedure LDET (4 arguments) computes the determinant of a matrix. Arguments are the matrix, the number of actual entries, the allocation dimension, and the determinant.
- Procedure MBLOCK (5 arguments) transforms a quadratic matrix to blocks on diagonal. Arguments are matrix, the transformation and its inverse, allocation and momentary dimension.
- Procedure SUBSTR (4 arguments) returns a substring. Arguments are string, first and last numbers identifying substring, and substring.
- Procedure STCRE (2 arguments) converts a string to a real. Argument are the string and the real.
- Procedure RECST (3 arguments) converts a real to a string using a FORTRAN format. Arguments are the real, the format, and the string.
- Procedure VELSET (3 arguments) sets a component of a vector of reals. Arguments are the vector, the number of the component, and the real value for the component to be set.
- Procedure VELGET (3 arguments) returns a component of a vector of reals. Arguments are the vector, the number of the component, and on return the real value of the component.
- Procedure VEZERO (3 arguments) used in repetitive tracking to prevent overflow due to lost particle.
- Procedure VELMAX (2 arguments) returns the maximum element of a vector.
- Procedure VEFILL (5 arguments) fills a vector array for scanning. Arguments: VE array, domain IN or IV, number of dimension, number of division, number of random points.

- Procedure IMUNIT (1 argument) returns the imaginary unit i.
- Procedure LTRUE (1 argument) returns the logical value true.
- Procedure LFALSE (1 argument) returns the logical value false.
- Procedure INTERV (3 arguments) produces an interval from 2 numbers. Arguments are the lower and upper bounds and on return the resulting interval.
- Procedure INSRND (1 argument) enables and disables outward rounding of intervals with the parameters 1 and 0 respectively. By default the rounding is enabled.
- Procedure INLO (2 arguments) returns the lower bound of an interval. Arguments are the interval and on return the lower bound.
- Procedure INUP (2 arguments) returns the upper bound of an interval. Arguments are the interval and on return the upper bound.
- Procedure IVVELO (2 arguments) returns the lower bounds of an interval vector as real vector.
- Procedure IVVEUP (2 arguments) returns the upper bounds of an interval vector as real vector.
- Procedure IVSET (3 arguments) sets a component of an interval vector. Arguments are the interval vector, the number of the component, and the interval for the component to be set.
- Procedure IVGET (3 arguments) returns a component of an interval vector. Arguments are the interval vector, the number of the component, and on return the interval of the component.
- Procedure OIELEM (2 arguments) returns first order ordered interval number. Arguments are the interval and the first order interval in return.
- Procedure OIIN (3 arguments) turns an ordered interval number into an interval by summing, neglecting the first terms to a given order. Arguments: interval, order, return.
- Procedure OVELEM (2 arguments) returns first order ordered interval vector. Arguments are an interval vector and the first order interval vector in return.
- Procedure OVIV (3 arguments) turns an ordered interval vector into an interval vector by summing, neglecting the first terms to a given order. Arguments: vector, order, return.

- Procedure INTPOL (2 arguments) determines coefficients of Polynomial satisfying $P(\pm 1) = \pm 1$, $P^{(i)}(\pm 1) = 0$, $i = 1, ..., n$. Arguments: coefficient array, n.

- Procedure RDAVAR (7 arguments) creates a RDA object. Arguments: a DA, NO, NV, reference point RE or VE, domain IN or IV, bounds for remainder and each order IN or IV, resulting RDA

- Procedure RDVAR (4 arguments) creates the i-th identity RDA. Arguments: i, reference point RE or VE, domain IN or IV, resulting i-th identity RDA.

- Procedure RDANOT (3 arguments) truncates a RDA to a lower order RDA. Arguments: RDA to be truncated, truncation order, resulting RDA.

- Procedure RDREA (2 arguments) reads a RDA. Arguments are the unit number and the variable name.

- Procedure DAEXT (2 arguments) extracts the DA part from a RDA. Arguments are the RDA and the resulting extracted DA vector.

- Procedure RDRBND (2 arguments) extracts the remainder bound interval from a RDA. Arguments are the RDA variable and the resulting remainder bound interval.

- Procedure RDAREF (2 arguments) extracts the reference points from a RDA. Arguments are the RDA variable and the reference point (in RE) or points (in VE).

- Procedure RDADOM (2 arguments) extracts the domain intervals from a RDA. Arguments are the RDA variable and the domain interval (in IN) or intervals (in IV).

- Procedure RDITIG (1 argument) sets the algorithm number of RDA tightening.

- Procedure RDNPNT (1 argument) sets the total number of points for scanning for RDA tightening with real rastering algorithms.

- Procedure RDINT (3 arguments) performs an integration of a RDA. Arguments are the number with respect to which to integrate, the RDA to be integrated and the resulting RDA.

- Procedure RDPRIS (1 argument) sets the RDA output option.

- Procedure CLEAR (1 argument) clears a graphics object.

- Procedure GRMOVE (4 arguments) produces a graphics object containing just one move. Arguments are the three coordinates and the graphics object.

- Procedure GRDRAW (4 arguments) produces a graphics object containing just one draw. Arguments are the three coordinates and the graphics object.
- Procedure GRDOT (4 arguments) produces a graphics object containing one move and a dot. Arguments are the three coordinates and the graphics object.
- Procedure GRCURV (10 arguments) produces a graphics object containing just one B-Spline. Arguments are the three final coordinates, the three components of the initial tangent vector, the three components of the final tangent vector and the graphics object.
- Procedure GRPROJ (3 arguments) sets 3D projection angles to a graphics object. Arguments are phi and theta in degrees and the graphics object.
- Procedure GRCHAR (2 arguments) produces a graphics object containing just one string. Arguments are the string and the graphics object.
- Procedure GRCOLR (2 arguments) produces a graphics object containing just one color change. Arguments are the new color id and the graphics object.
- Procedure GRWDTH (2 arguments) produces a graphics object containing just one width change. Arguments are the new width id and the graphics object.
- Procedure GRMIMA (7 arguments) finds the minimal and the maximal coordinates in a graphics object, arguments are the object and the minima and maxima for x, for y, and for z.
- Procedure RKCO (5 arguments) sets the coefficient arrays used in the COSY eighth order Runge Kutta integrator.
- Procedure POLVAL (7 arguments) performs POLVAL. See section 5.8 for details.
- Procedure POLSET (1 argument) sets the POLVAL algorithm number. 0: expanded, 1: Horner.
- Procedure FOXDPR (2 arguments) prints a dump of a variable. Arguments are the unit number and the variable name.
- Procedure MEMWRT (1 argument) writes memory to file : I, NBEG, NEND, NMAX, NTYP, CC, NC in first line and CC, NC in others. Argument is the unit number.
- Procedure VARMAX (1 argument) reports the maximum number of variables.

Index